Endbericht

PG 494 L2EE: Lightweight Process Coordination & J2EE

2. April 2007

Teilnehmer

Tim Bahlo	Christian Meves
Benjamin Bentmann	Stefan Naujokat
Philip Gerlach	Johannes Neubauer
Ilia Gorodnianski	Romuald Rasel
Benjamin Hanzelmann	Behrus Seyedi-Tabari
Darius Jockel	Stephan Windmüller

Betreuer

Prof. Dr. Bernhard Steffen
Dipl. Inform. Sven Jörges
Dipl. Inform. Ralf Nagel

Inhaltsverzeichnis

Teil I.

Einleitung

1. Projektbeschreibung

In diesem Kapitel wird ein erster Überblick der Absichten und Aufgaben der Projekgruppe 494 im Hinblick auf das zu realisierende Softwaresystem oCOBS vermittelt. Dieser allgemeinen Einführung in den Problembereich folgt in weiteren Kapiteln eine ausführliche Auseinandersetzung mit dem aktuell eingesetzten System und den verfügbaren Basistechnologien, bevor im nächsten Teil dieses Berichts auf die Details der Planung und Umsetzung eingegangen wird. Der letzte Teil schließt den Bogen zur Projektbeschreibung mit einer kritischen Gegenüberstellung von Soll- und Istzustand.

1.1. Motivation

Die an der Universität Trier betriebene Literaturdatenbank DBLP dient weltweit Personen bei der Recherche von Fachliteratur aus dem Bereich der Informatik. Ebenso werden mit Hilfe der DBLP und ihrer reichen Verweisstruktur zwischen Autoren und Publikationen Relevanzbewertungen vorgenommen.

Im Vergleich mit den diversen anderen Literaturdatenbanken sind zwei Besonderheiten der DBLP besonders zu betonen. Zum einen ist die DBLP nicht von Beginn an für ihre heutige Nutzung konzipiert worden. Was damals als einfaches Versuchsprojekt begann, ist – ausgelöst durch eine unerwartete Nachfrage seitens der ersten Benutzer – zu seiner heutigen Größe herangewachsen. Zum anderen legen die Betreiber der DBLP bei der Pflege des Systems das Hauptaugenmerk nicht auf die Quantität sondern auf die Qualität der erfassten bibliographischen Informationen. Nicht zuletzt diese Entscheidung trug und trägt zur Popularität der DBLP bei.

Erwartungsgemäß gibt es im Hinblick auf die Instandhaltung der DBLP zwei Hauptaufgaben: die Pflege und Erweiterung der Bibliographiedaten selbst und der Ausbau der Webanwendung und ihrer dargebotenen Funktionalität. Das Bestreben nach einem qualitativ hochwertigen und hochverfügbaren Service einerseits und die begrenzten finanziellen und personellen Mittel andererseits führten zu einer bevorzugten Wartung der Literaturdaten und damit einhergehend einer Vernachlässigung der zugrunde liegenden Anwendungsarchitektur.

Hier möchte die Projektgruppe unterstützen und mit ihrem *Dortmunder Online-Bibliographieservice*, kurz oCOBS, eine pflegeleichte und erweiterbare Plattform für den zukünftigen Betrieb der DBLP schaffen.

1.2. Ziele

Zur Realisierung einer erweiterbaren und vor allem zuverlässigen Anwendung sollen die diversen Funktionen der DBLP mit Hilfe des am Lehrstuhl 5 entwickelten Konzepts der *Lightweight Process Coordination* serviceorientiert nutzbar gemacht werden. Als Grundlage für die entstehende Webanwendung wird hierbei die *Java Enterprise Edition* (JEE) zum Einsatz kommen. Die von der Projektgruppe konkret angestrebten Ziele stellen sich wie folgt dar:

- Entwurf eines flexiblen Datenmodells, um spätere Erweiterungen an der Struktur der Literaturdaten ohne Änderung der Anwendung selbst realisieren zu können
- Umsetzung eines Importmoduls, mit dem die bestehenden Datenbestände der DBLP möglichst verlustfrei in das neue Datenmodell übertragen werden können
- Entwicklung einer Bibliothek von *Service Independent Building Blocks* (SIB), um bestehende und neue Funktionen der DBLP in wiederverwendbarer Form zu kapseln
- Bereitstellung einer rollenbasierten Benutzerverwaltung für die Mitarbeiter des Bibliographiedienstes, um den komplexen und mehrstufigen Prozess der Datenpflege angemessen unterstützen zu können
- Ausbau der Suchfunktionalität, damit Benutzer im stetig zunehmenden Bestand der DBLP schnell die gewünschte Information erhalten
- Erstellung einer ansprechenden Webschnittstelle mit deren Hilfe die Benutzer gewohnte und neue Dienste komfortabel nutzen können
- Integration einer Administrations- und Überwachungsanwendung zur vereinfachten Wartung des Gesamtsystems
- Ermöglichung ausgedehnter Lokalisierung der Benutzerschnittstellen durch Sprachenunabhängigkeit der Präsentationsschicht

2. DBLP

Die DBLP[1] in Trier ist die vielleicht einflussreichste Literaturdatenbank für Informatiker weltweit. Sie wird bei der generellen Literaturrecherche genauso konsultiert wie bei Begutachtungsprozessen im Rahmen von akademischen Bewerbern.

Im Datenbankbereich ist die DBLP der Universität Trier sogar die umfangreichste und wahrscheinlich einflussreichste Bibliographie. Ein Indiz dafür ist beispielsweise das häufige Erscheinen von Links zur DBLP auf der Suche nach Veröffentlichungen von Informatikern über Suchmaschinen wie *Google*.

2.1. Historie, Kennzahlen und Rollenverständnis

Im folgenden Kapitel wird eine kurze Beschreibung der Historie, eine Auswahl von Kennzahlen und das Rollenverständnis der DBLP vorgestellt.

2.1.1. Historie und Entwicklung

Seit Ende 1993 wird unter dem Namen DBLP ein Webserver für Fachinformationen aus dem Gebiet der Informatik an der Universität Trier betrieben, welcher zunächst eher als „Nebenprodukt“ aus der Doktorarbeit von Dr. Michael Ley entstand. Dies ist auch der Grund, warum zu Anfang nur Informationen zu den Themen „Datenbanksysteme“ und „Logikprogrammierung“ bereitgestellt worden sind [17]. Die positive Resonanz der Anwender hat die Betreiber jedoch ermutigt, den Informationsdienst auf weitere Teilgebiete der Informatik zu erweitern. So ist die DBLP von einer sehr spezialisierten kleinen Sammlung von bibliographischen Informationen zu einem bedeutenden Teil einer Infrastruktur, die von tausenden von Informatikern benutzt wird, herangewachsen.

Da die DBLP zu jeder Zeit ihrer Geschichte genutzt wurde und somit nicht etwa nur ein Forschungsprojekt war [17], sondern für eine Vielzahl von Benutzern einen Service angeboten hat, bestand und besteht immer noch ein Konflikt zwischen der Entwicklung neuer Features sowie dem Einsatz neuer Softwarekomponenten und der Instandhaltung und Aktualisierung bereits vorhandener Daten unter der Beachtung hoher Qualitätsansprüche sowie dem Hinzufügen neuer Inhalte mit hoher Qualität.

[1] früher: „DBLP“ = DataBase systems and Logic Programming, seit der Inhaltserweiterung: „DBLP“ = Digital Bibliography & Library Project

Unterstützt durch die Erfahrung, dass die Instandhaltung und Pflege von komplexen Softwaresystemen mehr Zeit in Anspruch nimmt, als der durch diese Software entstehende Vorteil an Gewinn einbringt, haben sich die Betreiber in den meisten Fällen für die stärkere Beachtung des zweiten Punktes und somit gegen neue Softwarekonzepte entschieden. Weiterführende Informationen, welche die aus diesen Entscheidungen resultierende Architektur und Software der DBLP betreffen, sind in Kapitel 2.4 auf Seite 10 zu finden.

2.1.2. Kennzahlen

Die DBLP bietet Informationen über mehr als 730.000 Artikel[2] an.

Durch die Integration der *ACM SIGMOD Anthology* (siehe hierzu [5]) in die DBLP sind über 100.000 *citation links* verfügbar. Das heißt, zu einer Reihe von Artikeln sind jeweils die referenzierten und referenzierenden Artikel verlinkt.

Außerdem enthält die DBLP mehrere tausend Links zu den Internetseiten von Informatikern, die Fachinformationen publizieren.

2.1.3. Rollenverständnis

Die Betreiber der DBLP betrachten diese als einen *Bibliographie-Server light* [16]. Der Server speichert keine Dokumente, sondern nur Informationen über diese. Hier spielt die DBLP nur die Rolle eines Vermittlers von Metainformationen und liefert neben den anderen Serviceangeboten – wie beispielsweise der Abbildung des komplexen sozialen Netzwerks der Autoren – lediglich einen Verweis zum Serviceanbieter, von welchem letztendlich die Publikationen abgerufen werden können.

Zunächst finanzierte sich die DBLP durch eine Reihe von Preisgeldern, die Michael Ley für seine Arbeit erhalten hat. Mittlerweile wird der Bibliographie-Server von mehreren Sponsoren unterstützt. Nennenswert sind hier *ACM SIGMOD* [1] und Rick Snodgrass sowie *Microsoft Bay Area Research Center* [19] und Jim Gray.

2.2. Generelles zu Artikeln

Um mit *Fachinformationen* zu arbeiten beziehungsweise Dienste über diese anbieten zu können, ist es wichtig zu wissen, wie diese Informationen aufgebaut sind und über welche Medien diese normalerweise publiziert werden. Außerdem sollte natürlich auch über diese Medien jegliches „strukturierungsrelevante“ Wissen (Struktur, Erscheinungsform, Dateiformat...) vorhanden sein.

Erweiterte Suchdienste oder Darstellungen und somit ein leichter Zugang zu Fachinformationen können also nur sinnvoll entwickelt und erstellt werden, wenn bekannt

[2] Stand März 2006

ist, wie das entsprechende „Rohmaterial“ aussieht und wie es strukturiert ist. In diesem Kapitel werden einige besonders wichtige Aspekte dieses Wissens vorgestellt. Eine ausführlichere Zusammenstellung findet sich beispielsweise in [5].

2.2.1. Publikationen

In der Informatik wird nach wie vor ein großer Teil qualitativ hochwertiger Fachinformation über konventionelle Medien wie Zeitschriften und Tagungsbände verbreitet [16].

Man unterscheidet die folgenden Formen von *Publikationen*:

- Monographie
- Artikel in Fachzeitschriften
- Artikel in Konferenzbänden
- Artikel in Sammelbänden

Bibliotheken (an Hochschulen oder großen Forschungseinrichtungen) haben einen Teil dieser Publikationen als ganze Bände oder einzelne Zeitschriftentitel in Katalogen (*OPACs*) in ihrem Bestand, so dass Bibliotheksbenutzer lokalen Zugriff darauf haben. Auf viele dieser Kataloge ist über das Internet auch bereits ein externer Zugriff möglich. Einzelarbeiten aus Zeitschriften, Tagungs- und Sammelbänden werden in OPACs nicht aufgeführt. Um auf diese Arbeiten zugreifen zu können, muss daher auf externe Bibliographien (beispielsweise den auf Papier und CD-ROM publizierten *ACM Guide to Computing Literature*[3]) oder kostenpflichtige Datenbanken (beispielsweise *CompuScience*[4]) zurückgegriffen werden. Diese Dienste weisen jedoch eine Vielzahl von Nachteilen auf, welche die Anzahl der Zugriffe relativ stark einschränken. Zu nennende Nachteile sind unter anderem zu hohe Kosten für den Zugriff, mangelnde Aktualität und mangelnde Gesamtheit der hinterlegten Artikel.

Die Struktur und der Aufbau unterschiedlicher naturwissenschaftlicher Artikel sind in der Regel immer gleich beziehungsweise folgen einem gleichen Schema. Eine ausführliche Beschreibung der Gliederung eines naturwissenschaftlichen Artikels ist in [5] zu finden.

[3] Dieses jährlich vom *ACM* herausgegebene Nachschlagewerk schlüsselt in Zeitschriften oder Monographien erscheinende Informatik-Literatur auf, zum Teil sind sogar Technical Reports aufgeführt.

[4] Enthält weltweite Publikationen auf den Gebieten der Informatik und Computertechnologie. Der Datenbestand umfasst derzeit über 500.000 Publikationen von 1972 bis heute und wird monatlich mit ca. 2.500 Einträgen aktualisiert. Reports, Dissertationen und Preprints werden dabei ebenfalls berücksichtigt. [8]

2.2.2. Rechtliches

Die rechtliche Situation bei der Veröffentlichung von Fachinformationen über das Internet ist in Bezug auf Urheberrechte und wissenschaftliche Kommunikation problematisch und widersprüchlich [16]. Eine Vielzahl von Artikeln steht im Web als Volltext zur Verfügung. Teilweise handelt es sich dabei um „Preprints“[5]. Sehr viele der Artikel sind jedoch bereits in Zeitschriften oder Tagungsbänden publiziert worden. Verleger verlangen in der Regel vor der Publikation von Artikeln die Übertragung der Urheberrechte und verbieten die Verbreitung über das Internet. Diese Rechtslage wird jedoch von vielen Autoren ignoriert. Sie bieten „ihre“ Arbeiten auf ihrer Homepage zum Download an. Dabei steht meistens kein wirtschaftliches Interesse im Vordergrund. Der Grund ist oft der Wunsch nach einer möglichst weiten Verbreitung der Arbeiten. Außerdem wird das systematische Sammeln von Verweisen auf Online-Arbeiten durch die Verleger verboten. Bisher ist nach [16] nicht bekannt, inwiefern die Verlage Gebrauch von ihrem Urheberrecht machen und dies mit juristischen Schritten durchsetzen. Vorstellbar wären beispielsweise Klagen gegen Autoren, die „ihre“ Arbeiten im Internet anbieten.

Der Aufbau von brauchbaren digitalen Bibliotheken erscheint durch die gegebene rechtliche Situation beinahe unmöglich. Hierbei stehen das Bedürfnis nach freier Kommunikation, die Notwendigkeit wissenschaftliche Resultate zuverlässig zu archivieren und die wirtschaftlichen Interessen der Verleger in Konflikt.

2.2.3. Häufig benutzte Abkürzungen und Begriffe

Im Zusammenhang mit Fachinformation gebräuchliche Abkürzungen und Begriffe sind:

TOC **T**able **O**f **C**ontents

OPAC **O**nline **P**ublic **A**ccess **C**atalogue (deutsch: Bibliothekskatalog)

Proceedings Konferenzbänder

Inproceedings Artikel innerhalb eines Konferenzbandes

citation links Liste von Links zu den referenzierten und referenzierenden Artikeln einer Publikation

2.3. Benutzerzugriff auf die Bibliographie

Im Folgenden wird die DBLP aus der Sicht des Nutzers, das heißt, so wie der Benutzer die Struktur, Informationen und Funktionen wahrnimmt, dargestellt. Details

[5] Artikel, die zur Veröffentlichung eingereicht wurden

zu Softwareprinzipien und Realisierungskonzepte sind Kapitel 2.4 auf Seite 10 zu entnehmen.

2.3.1. Oberfläche

Die Startseite der DBLP ist in mehrere Bereiche aufgeteilt. Der Benutzer hat unterschiedliche Möglichkeiten, von hier aus auf die Daten der DBLP zuzugreifen und zu anderen Informationen über die gelisteten Artikel zu gelangen. Des Weiteren kann er zu den unterschiedlichen Spiegelungen der Datenbank gelangen und auf eher allgemeine Informationen über die Bibliothek (Erläuterung zum Umgang mit DBLP, FAQs, Link zur Homepage von Michael Ley) zugreifen. Neben den eigentlichen Zugriffsmöglichkeiten auf die Bibliothek, welche in den nachfolgenden Abschnitten erläutert werden, bietet die Startseite eine Reihe von nützlichen Links und die aktuellen Neuigkeiten zur DBLP an. Außerdem werden die beteiligten Organisationen und Förderer genannt.

2.3.2. Suche

Um Informationen in der DBLP zu finden, werden drei unterschiedliche Möglichkeiten der Suche bereitgestellt:

Nach Autor

In ein Textfeld kann der Benutzer den Namen eines Autors eingegeben. Ist der eingegebene Name mehrmals vorhanden, werden alle Treffer angezeigt und aus einer Liste kann der gewünschte Name gewählt werden. Ist der Name nur einmal vorhanden, wird direkt die Seite der Person angezeigt.

Zu jedem der Bibliothek bekannten Autor wird so schließlich eine Seite mit dem Suchergebnis gefunden.

Jedes Ergebnis enthält jeweils in dieser Reihenfolge folgende vier Komponenten:

- Eine Reihe von Links zu anderen Suchmöglichkeiten (siehe hierzu auch Kapitel 2.3.4 auf Seite 10),
- einen Link zur Homepage des Autors (falls bekannt),
- eine Auflistung der Arbeiten des Autors sowie
- eine Liste der Ko-Autoren (*Ko-Autor-Index*).

Die Auflistung der Arbeiten enthält, neben einem Link zu der Publikation, jeweils die Liste aller Autoren, den Titel der Publikation sowie eventuell den Namen (inklusive Verlinkung) des Konferenzbandes, der Zeitschrift oder des Verlags.

Der Link zur Homepage des Autors kann in vielen Fällen hilfreich sein, um sich beispielsweise ein besseres Bild vom Autor zu machen, ein Foto und einen Lebenslauf zu finden oder auch um insbesondere die aktuellsten Forschungsergebnisse und Publikationen nachlesen zu können.

Beim *Ko-Autor Index* handelt es sich um eine alphabetische Auflistung aller mit dem Autor zusammen arbeitenden Autoren. Der Betrachter kann somit direkt in das komplexe soziale Netzwerk hineinsehen, in welchem jeder Autor sich wiederfindet.

Bei der Suche nach einem Autor wird lediglich ein einfacher Algorithmus, der Teilzeichenfolgen sucht, angewandt. Bei der Eingabe von Großbuchstaben werden an dieser Stelle auch nur Großbuchstaben gefunden, bei Kleinbuchstaben, sowohl Klein- als auch Großbuchstaben. „ley“ würde also Stanley, Kley, Ley, Leyman... finden, wohingegen „Ley“ nur Ley, Leyman... finden würde. Werden außergewöhnliche Buchstaben wie zum Beispiel ä, ü, ß... eingegeben, so werden auch Ergebnisse mit den zugehörigen „einfachen“ Buchstaben gefunden. Gemeint ist hiermit beispielsweise: „ss“ für „ß“, „u“ für „ü“, „a“ für „ä“ usw. Enthält das Suchwort in HTML-Code beschriebene Sonderzeichen (also beispielsweise `ü` oder `á`)[6], so werden die dem Code entsprechenden Ergebnisse gefunden, allerdings ohne die oben beschriebenen Ergebnisse mit „einfachen“ Buchstaben. *M&uumller* findet also Müller, aber nicht Muller.

Nach Titel

Bei der Suche nach Artikeln mittels Eingabe des Titels wird dem Benutzer wiederum ein schlichtes zu befüllendes Textfeld angeboten. Das Suchergebnis zeigt alle zu dem Suchwort passenden Publikationen jeweils mit Autorenliste, komplettem Titel und Publikationsort. Der Benutzer kann dann über Links zu den Autorenseiten oder zu den Publikationsorganen gelangen. Von dort sind die Informationen über die Artikel erreichbar. Die Unterscheidung zwischen Klein- und Großbuchstaben gilt genauso wie bei der Suche nach Autoren.

Erweiterte Suche

Die erweiterte Suche lässt neben der Eingabe von mehreren Autoren (mit logischem „und“ verknüpft) und dem Titel das Eintippen des Erscheinungsjahrs, der Seitennummer und der ID zu. Zusätzlich kann je nach Publikationsform entweder der Name der Konferenz oder der Name, der Jahrgang und die Nummer des Journals angegeben werden.

[6] `ü` für ü; `á` für á

2.3.3. Bibliographien

Neben der Suche nach Artikeln über bestimmte Suchkriterien ist der Zugriff auf Publikationen über die Inhaltsverzeichnisse der einzelnen Publikationsorgane möglich.

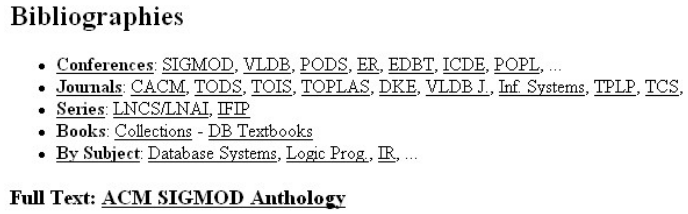


Abbildung 2.1.: Ausschnitt der Startseite der DBLP: Zugriff auf Artikel über Publikationsorgane und Themengebiete oder über die *ACM SIGMOD Anthology*

Über *Conferences*, *Journals*, *Series* und *Books* kann der Benutzer auf eine Liste der jeweils verfügbaren Publikationsorgane zugreifen (siehe Abbildung 2.1).

Wählt er beispielsweise eine Zeitschrift, so kann anschließend der Band und die Nummer ausgewählt werden, woraufhin die Liste der zugehörigen Artikel angezeigt wird.

Außerdem steht eine Kategorisierung der Publikationsorgane nach Themen (siehe Abbildung 2.1: „By Subject“) zur Verfügung, wodurch ein weiterer Zugang zu den Artikeln möglich wird. Wählt man ein bestimmtes Themengebiet, werden alle Zeitschriften, Konferenzen angezeigt, die sich mit dem gewählten Thema befassen. Der weitere Zugriff erfolgt dann wieder „direkt“ über die jeweilige Publikationsform.

2.3.4. Links

Im Bereich *Links* der Startseite findet der Benutzer eine Sammlung von Links zu unterschiedlichen Organisationen. Es handelt sich hierbei um Forschungsgruppen und offizielle Informatikorganisationen. Des Weiteren sind Links zu *serviceverwandten Organisationen* der DBLP aufgeführt. Diese können bei der Suche nach Fachinformationen alternativ oder als Ergänzung aufgesucht werden.

In [5] werden zwei dieser *serviceverwandten Organisationen* kurz vorgestellt und nach unterschiedlichen Aspekten mit der DBLP verglichen.

2.4. Aufbau und Realisierung des DBLP-Servers

Im Folgenden soll die grundlegende Strukturierung beschrieben werden, die hinter dem in Kapitel 2.3 auf Seite 7 beschriebenen Benutzerzugriff auf die DBLP steht. Es wird der Einsatz von Softwarekonzepten erläutert, welche den zur Verfügung stehenden Service ermöglichen.

2.4.1. Grundlegende Strukturen

Die grundlegende Basisstruktur der DBLP wird in Abbildung 2.2 illustriert. Auf der Startseite kann die Gruppe der Konferenzen oder Zeitschriften ausgewählt werden. Von dort aus sind die einzelnen Elemente dieser Gruppen erreichbar. Da Konferenzen und Zeitschriften in der Regel in bestimmten zeitlichen Abständen stattfinden beziehungsweise herausgegeben werden, wird in der nächsten Stufe zu den jeweiligen Instanzen der Konferenz oder Zeitschrift verzweigt. Diese enthalten dann Inhaltsverzeichnisse der zugehörigen Artikel.

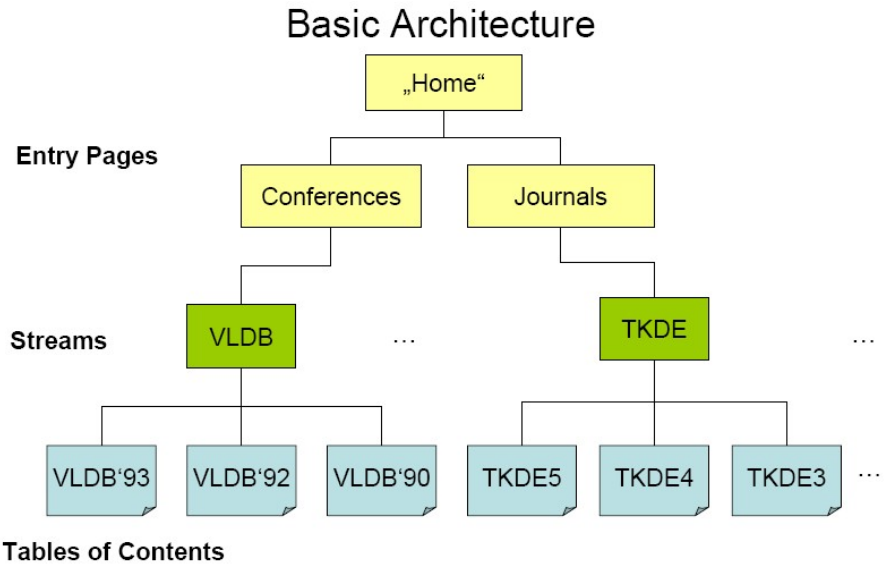


Abbildung 2.2.: Die grundlegende Struktur der DBLP [17]

Neben der beschriebenen Struktur wird über den bekannten Instanzen der Veröffentlichungen ein *Personen-Publikations-Netzwerk* aufgebaut (siehe Abbildung 2.3 auf der nächsten Seite). Dabei wird zu dem Namen jeder Person eine Seite aufgebaut, zu der alle Publikationen der Person verlinkt sind. Dies sind die in Kapitel 2.3.2 auf Seite 8 beschriebenen Personenseiten. Datenbanktechnisch handelt es sich dabei nur um eine Ansicht auf die Datenbank und wird nach eigenen Informationen [17] „a 'canned' query“[7] genannt.

2.4.2. Datenbankmanagementsystem

Bis heute wird in der DBLP kein *DBMS*[8] verwendet. Im Rahmen von zwei Diplomarbeiten in den Jahren 1996 und 1998 ist mit unterschiedlichen Ansätzen theoretisch nutzbare und funktionierende Prototyp-Datenbank-Software für die DBLP

[7] Übersetzung ungefähr: „eine fest gespeicherte Suchanfrage“
[8] DBMS = **D**ata**b**ase **M**anagement **S**ystem

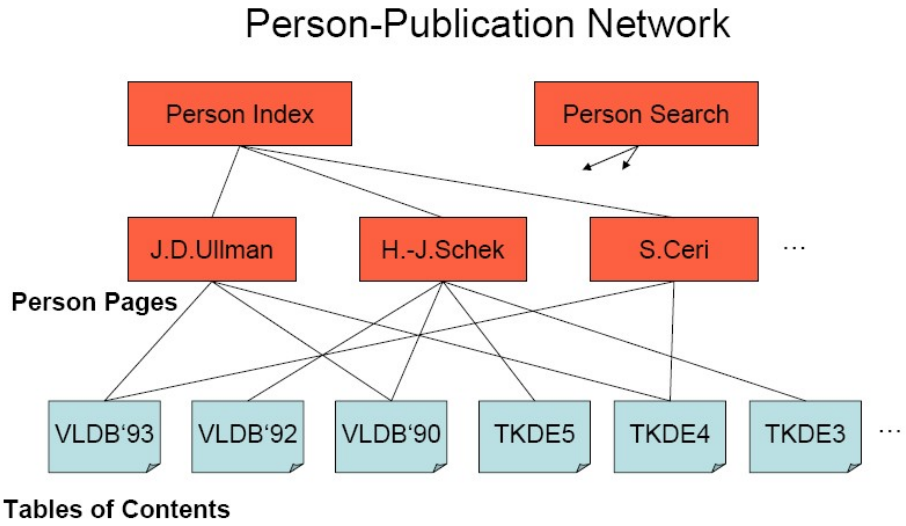


Abbildung 2.3.: Idee des *Personen-Publikations-Netzwerks* der DBLP [17]

entwickelt worden. Obwohl durch die Benutzung eines DBMS bessere Werkzeuge bereitgestanden hätten, um die Konsistenz der Daten zu wahren, überwog für die Betreiber der DBLP der Nachteil, dass es zu zeitaufwändig gewesen wäre, diese Software wirklich zum Einsatz zu bringen.

2.4.3. Softwarestrategien und Komponenten

Das Format und die Weiterverarbeitung der gespeicherten Datensätze sind entscheidend für die effiziente Realisierung der bereitgestellten Dienste. In den folgenden Abschnitten wird die Art der Speicherung der bibliographischen Datensätze beschrieben. Außerdem wird darauf eingegangen, mit Hilfe welcher Softwarestrategien die einzelnen Dienste angeboten werden.

HTML TOCs und Author Pages

Die im Anfangsstadium vorhandenen per Hand erstellten *HTML TOCs* besitzen eine sehr homogene Struktur. Dieses standardisierte Format macht das Parsen besonders einfach. Um die in Kapitel 2.3.2 auf Seite 8 beschriebenen Autorenseiten zu erstellen, wurde bei der ursprünglichen Verarbeitung in zwei Schritten vorgegangen.

1. Parsen aller *TOCs* und Erstellen einer großen „nur-Text“-Datei *TOC_OUT*.

2. Erstellen der Autorenseiten mit Hilfe des – von Herrn Ley entwickelten – Programms *mkauthors*.

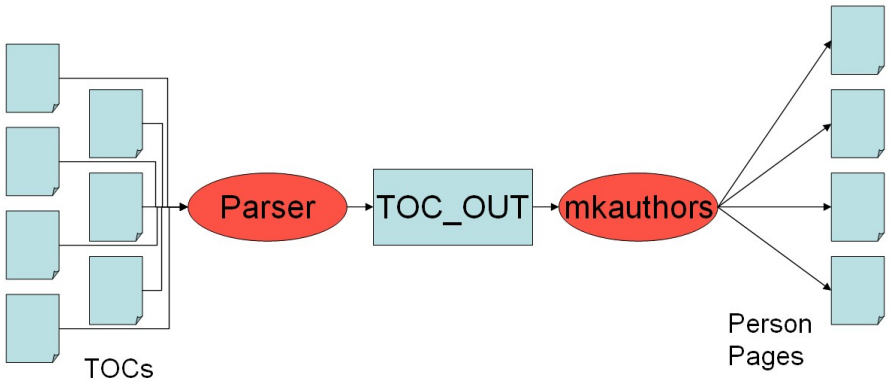


Abbildung 2.4.: TOC-Parser: Schritte zur Erstellung der Autorenseiten [17]

Abbildung 2.4 auf der nächsten Seite illustriert dieses schrittweise vorgehende Verfahren.

Beim *Parsen* im ersten Schritt werden alle wichtigen Informationen aus dem HTML-Text gelesen und in die *TOC_OUT-Datei* geschrieben. Das Programm *mkauthors* liest diese Datei anschließend in eine kompakte Hauptspeicher-Datenstruktur und generiert alle Autorenseiten, sowie eine Liste aller Autorenseiten. Zudem wird die Datei *AUTHORS*, welche alle Autorennamen enthält, erzeugt.

Auf allen Autorenseiten und Ergebnisseiten der Suche nach Titeln (Kapitel 2.3.2 auf Seite 9) sind die Namen der Autoren mit den jeweiligen Autorenseiten verknüpft. Um diese Links in die ursprünglichen *TOCs* einzufügen, wird eine modifizierte Version des Parsers ausgeführt. Dieses Programm fügt in allen *TOC*-Dateien die jeweiligen Links hinzu, wenn die Autorenseiten verfügbar sind. Die Ausführung des Programms erfolgt nach jeder Ausführung von *mkauthors*.

Es wäre denkbar und aus unterschiedlichen Gründen vermutlich sinnvoll, die Autorenseiten nur auf Anfrage zu erstellen. Die Betreiber haben sich allerdings dazu entschlossen, das Programm *mkauthors*, sofern Änderungen an den Daten vorgenommen wurden, jeden Abend auszuführen und somit alle Autorenseiten neu zu erzeugen. Es wurde beschlossen, dass es einfacher und billiger sei, den notwendigen Speicherplatz zur Verfügung zu stellen, als einen Index zu generieren und jede Seite nur bei Bedarf zu generieren.

XML Records

Es ist offensichtlich, dass HTML-Tabellen als primärer Speicherort der bibliographischen Daten nicht die in jedem Fall beste Lösung sind [17]. Die Integration von *reading lists*[9] für Seminare und Kurse, kommentierte Bibliographien, Bewer-

[9] *reading lists* = Leselisten; bspw. für Vorlesungen empfohlene Literaturlisten

tungen und *citation linkings*[10] machen es notwendig, jeder Publikation eine eindeutige ID zuzuweisen (und sie darüber erreichbar zu machen). Hierzu ist es erforderlich, Informationen in klassischeren bibliographischen Datensätzen zu speichern. Das Standardformat *BibTeX* für bibliographische Datensätze der Informatik ist jedoch nicht einfach zu parsen. Die Betreiber der DBLP haben sich dazu entschlossen, die bibliographischen Datensätze aus *BibTeX* in XML-Syntax darzustellen. Für jede Veröffentlichung wurde so aus den ursprünglichen *TOCs* eine kleine Datei mit den wesentlichen Daten erzeugt. Auf dem Hauptrechner der DBLP existiert für jeden bibliographischen Datensatz eine eigene Datei. Als Schlüssel zum Zugriff auf den Datensatz wird der Dateiname benutzt. Abbildung 2.5 zeigt ein Beispiel für einen solchen Datensatz.

```
<article key="journals/algorithmica/NavarroB01">
  <author>Gonzalo Navarro</author>
  <author>Ricardo A. Baeza-Yates</author>
  <title>Improving an Algorithm for Approximate
      Pattern Matching.</title>
  <pages>473-502</pages>
  <year>2001</year>
  <volume>30</volume>
  <journal>Algorithmica</journal>
  <number>4</number>
  <ee>http://link.springer.de/link/service/journals/00453/
    contents/01/0034/</ee>
  <url>db/journals/algorithmica/algorithmica30.html#NavarroB01</url>
</article>
```

Abbildung 2.5.: Beispiel für einen DBLP XML Datensatz [17]

Unter [15] ist die *DTD* der XML-Dateien sowie die komplette DBLP-XML-Datei verfügbar.

BHT (Bibliography Hypertext)

Unstrukturierte Kommentare und Anmerkungen eignen sich nicht für die Integration in die XML-Datensätze. Sie erfordern in der DTD viele neue, aber wenig genutzte Felder. Das gleiche gilt für einige weitere in den *TOCs* vorhandenen Daten, beispielsweise Querverweise zwischen HTML-Seiten und *citation links*[11] usw. Es wurde deshalb beschlossen, solche Informationen weiterhin in den *TOCs* in *BHT-Dateien*

[10] Das Einfügen von *citation linkings*, um die im Literaturverzeichnis aufgelisteten Arbeiten mit den zugehörigen bibliographischen Datensätzen erreichen zu können.

[11] In der Bearbeitungsphase der *ACM SIGMOD Anthology* sind hier jedoch trotzdem einige Links in die XML-Daten aufgenommen worden.

zu speichern. Die XML Datensätze werden dann mit einem einfachen „include“-Mechanismus in die *TOCs* gelesen. Das Programm *mkhtml*, ein HTML-Präprozessor, liest eine *BHT-Datei* und ersetzt jedes `<cite key=“...“ style=“...“>` durch die in XML gespeicherten bibliographischen Informationen der DBLP. `cite key` definiert dabei den Speicherort beziehungsweise die ID des Datensatzes. `style` bestimmt das Format. Das Verfahren entspricht in etwa dem von `cite{...}` in LaTeX. *mkhtml* kann außerdem Logos und Fußnoten in die entstehenden HTML-Seiten einfügen. Es werden wieder (wie bei den Autorenseiten vgl. Kapitel 2.4.3 auf Seite 12) alle HTML-Seiten erzeugt. Dadurch wird zwar mehr Speicherplatz, aber weniger Bandbreite, benötigt. Abbildung 2.6 stellt dar, wie aus den bibliographischen Datensätzen und einer *BHT-Datei* die *TOCs* erstellt werden.

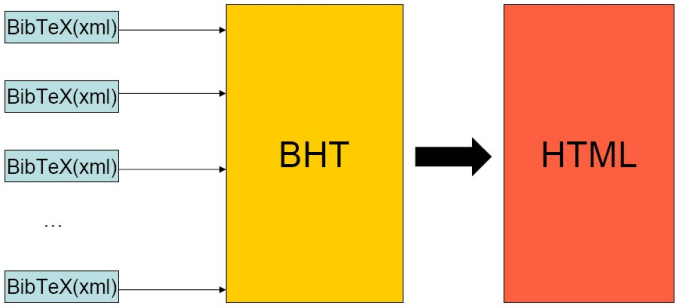


Abbildung 2.6.: Verfahren zur Erzeugung der *TOCs* aus bibliographischen Datensätzen und *BHT-Datei* [17]

Einfache Suche

Die in Kapitel 2.4.3 auf Seite 12 beschriebenen Dateien *TOC_OUT* und *AUTHORS* dienen als Eingabe für die Suche nach Titeln/Autoren (Kapitel 2.3.2 auf Seite 8/ 2.3.2 auf Seite 9). Diese Suche wird durch die in *C* geschriebenen Suchprogramme *author* und *title* ermöglicht. Die Programme arbeiten „brute force“[12], das heißt, sie führen für jede Anfrage eine sequentielle Suche durch. Ein wichtiger Vorteil dieser primitiven Suchtechnik ist die Tatsache, dass die Installation auf gespiegelten Servern keine Probleme bereitet.

Erweiterte Suche

Für die Suchmaschine der erweiterten Suche wird Software der Firma *MG* benutzt, welche in [35] beschrieben wird. Hierbei handelt es sich um eine Veröffentlichung, die an der Universität Melbourne entstanden ist.

[12] „brute force“ = rohe Gewalt

Die XML Datensätze werden in das *MG Suchsystem* geladen. Suchergebnisse dieses „erweiterten Suchsystems“ werden mit auf CGI basierenden Web-Schnittstellen in einem den Autorenseiten ähnlichem Stil dargestellt. Die MG Software behandelt dabei jedes XML Dokument zunächst als ein Textdokument. Die Feldstruktur wird erst nachher erkannt. Daraus resultierende falsche Suchergebnisse werden erst in einem zweiten Schritt herausgefiltert.

2.4.4. Einpflegen neuer Daten

Das *Einpflegen neuer Daten in die DBLP* geschieht prinzipiell nur im Rahmen von kompletten Publikationssammlungen. Die Aufnahme von Einzelarbeiten wird unter anderem aus Zeitgründen nicht unterstützt. Die Betreiber sind außerdem der Ansicht, dass eine öffentliche HTML-Seite zur Einstellung neuer bibliographischer Datensätze zu häufig missbraucht würde. Es wird jedoch darauf hingewiesen, dass Vorschläge für die Aufnahme von bestimmten Zeitschriften oder Konferenzen per E-Mail durchaus erwünscht sind.

3. Verwendete Technologien

3.1. Java Enterprise Edition

Die *Java Enterprise Edition* stellt eine Plattform für verteilte, mehrschichtige Anwendungen auf Basis der Programmiersprache Java dar. Dazu erweitert die Java EE nicht den Umfang der Sprache, wie sie in der *Java Standard Edition* definiert ist, sondern beschreibt Spezifikationen, Schnittstellen und Technologien, welche die Entwicklung und Ausführung von skalierbaren, modularisierten und robusten Mehrschicht-Anwendungen vereinfachen und erleichtern sollen. Die Ausführung dieser Anwendungen findet in der Regel in der Laufzeit-Umgebung eines sogenannten Application Servers (siehe Kapitel 3.6 auf Seite 25) statt.

Das Ziel der Vereinfachung wurde jedoch in den ersten Versionen der Java EE nur teilweise erreicht. Hierfür stellte die *Enterprise JavaBean* (EJB) Technologie vor der Version 3 ein vielfach kritisiertes Negativ-Beispiel dar. So entstanden zahllose Frameworks und Tools, wie zum Beispiel *Struts*, *Spring* oder *Hibernate*, die ähnliche Funktionen wie die Java EE anbieten und dabei größeren Wert auf eine einfache Handhabung legen.

3.1.1. Java EE 5

Unter anderem als Reaktion auf diese Konkurrenzprodukte wurde am 11. Mai 2006 die *Java Enterprise Edition 5* veröffentlicht, welche einige Technologien zur Verringerung der Komplexität beinhaltet, die auch in der Projektgruppe genutzt werden. Darunter befinden sich zum Beispiel:

- Verwendung von – in der *Java SE 5* neu eingeführten – Annotationen für Konfigurationszwecke innerhalb des Quelltextes;
- *Enterprise JavaBeans* in der Version 3, die eine starke Verringerung der Komplexität und der Programmier-Artefakte zur Erstellung einer EJB im Vergleich zu den Vorgängerversionen ermöglicht;
- neuer Persistenzmechanismus für die Entity EJBs (siehe Kapitel 3.2 auf Seite 19), der die Integration von bestehenden ORM[1]-Lösungen wie *Hibernate* oder *TopLink* erlaubt;

[1] ORM - **O**bject-**R**elational **M**apping

- Verringerung der Abhängigkeiten zwischen Objekten durch Anwendung des *Dependency Injection* Entwurfsmusters.

Im Rahmen der Veröffentlichung der Java Enterprise Edition 5 wurde auch deren Abkürzung von *J2EE* in *Java EE* geändert, so dass die Bezeichnung der Projektgruppe leider nicht mehr konsistent mit der aktuellen Nomenklatur von *Sun* ist.

3.1.2. Verwendung in der Projektgruppe

Die im Rahmen der Projektgruppe entwickelte Anwendung besteht im Wesentlichen aus folgenden vier Schichten:

- Auf der untersten Ebene befindet sich die *Datenschicht*. Diese besteht aus einer – vom Java Application Server unabhängigen – relationalen Datenbank und dem EJB-Persistenzmechanismus, der die Datenbank-Tabellen als Entity EJBs kapselt.
- Über der Datenschicht liegt die *Logikschicht*, die die Verarbeitungslogik der Anwendung in Form von Session EJBs beinhaltet.
- Gemäß *Lightweight Process Coordination* wird zwischen der Logik- und der Präsentationsschicht die Koordinationsschicht eingefügt, welche im Kapitel 3.7 auf Seite 27 beschrieben wird.
- Zuoberst befindet sich die *Präsentationsschicht*, die unter Nutzung der unteren Schichten für die Interaktion mit dem Benutzer verantwortlich ist. Als Technologien zur Implementierung dieser Schicht werden einerseits die *JavaServer Pages* (siehe Kapitel 3.4 auf Seite 23) für den öffentlichen Webclient als auch die *JavaServer Faces* (siehe Kapitel 3.5 auf Seite 24) für die Administrationsoberfläche verwendet.

Als Schnittstelle zwischen Präsentationsschicht auf der einen und Logik- und Koordinationsschicht auf der anderen Seite fungieren im Fall der Administrationsoberfläche *Web Services* (siehe Kapitel 3.3 auf Seite 21), die eine XML-basierte Kapselung der Session EJBs darstellen und somit auch nicht-Java-Anwendungen erlauben, darauf zuzugreifen.
Der Webclient arbeitet hingegen direkt mit den Schnittstellen der jeweiligen Session-EJBs und erhält die jeweiligen Objekte mittels des *Java Naming and Directory Interfaces* (JNDI), einer API für einen Verzeichnisdienst, über den Ressourcen und verteilte Objekte in einem Kontext abgelegt und dann an anderen Stellen in der Anwendung wieder abgerufen werden können.

Der Aufbau der Anwendung als Vier-Schichten-Architektur erlaubt eine Verteilung der einzelnen Schichten auf unterschiedliche Server sowie Clustering innerhalb

der Schichten, so dass bei entsprechender Programmierung eine gute Skalierbarkeit der Anwendung gewährleistet ist.

3.2. Hibernate

Fast jede Anwendung braucht eine zuverlässige Methode, um ihre Daten zu speichern. Dafür gibt es viele Lösungen, sowohl einfache als auch komplexe. Eine davon ist *Hibernate*[20][10].

Der Nachteil der einfachen Lösungen ist, dass sie nicht so mächtig sein können wie größere, kompliziertere. Jede zusätzliche Fähigkeit erhöht die Komplexität des Ganzen. Hibernate versucht, einen Kompromiss zu finden zwischen der Leichtigkeit und Benutzerfreundlichkeit einer einfachen und der Mächtigkeit einer komplexen Lösung.

3.2.1. Was ist Hibernate?

Hibernate übersetzt zwischen der objektorientierten Sichtweise von Java auf der einen und der relationalen Sicht einer Datenbank auf der anderen Seite.

Hibernate bietet die Möglichkeit, viele der bei der dauerhaften Speicherung von Objekten anfallenden Tätigkeiten zu automatisieren. Unter anderem verwaltet es Datenbankverbindungen und Caches zur Laufzeit und kümmert sich um das Speichern von Werten und Objektreferenzen. Weiterhin kann es zum Beispiel Datenbankschemata bei der Installation der Anwendung erzeugen und an die Datenbank übertragen.

Um Hibernate einzusetzen, müssen keine schwerwiegenden Änderungen am Quellcode durchgeführt und auch keine Namensregeln oder Ähnliches beachtet werden. Es werden lediglich einfache „Plain Old Java Objects“, kurz „POJOs“, benötigt.

Ein POJO ist eine Java-Klasse, die ohne weitere Anpassungen persistent gemacht werden kann. Die Klasse muss lediglich einen Standardkonstruktor ohne Argumente enthalten. Selbst dieser kann `private`- oder `package`-sichtbar sein. Dadurch müssen Methoden, die nur für den Persistenzmechanismus notwendig sind, nicht öffentlich in die API aufgenommen werden.

3.2.2. EJB2 und Hibernate

Entity Beans wurden dazu entworfen, Daten in einer Datenbank zu speichern, zu laden und damit zu repräsentieren. Welche Vorteile bietet Hibernate?

Es gibt zwei Arten von Entity Beans: *BMP* (Bean Managed Persistence) und *CMP* (Container Managed Persistence).

Bei *BMP* ist die Bean selbst dafür verantwortlich, dass ihre Daten gespeichert werden. Der Programmierer der Bean muss also unter anderem direkt einen Daten-

banktreiber (zum Beispiel JDBC) benutzen und damit eine Datenbankverbindung aufbauen und auch darauf achten, dass der Datenbankbestand konsistent bleibt, wenn ein Fehler auftritt.

CMP verlagert die Datenspeicherung von der Bean in den Container. Das ist eine strukturelle Verbesserung gegenüber *BMP*, da zum Beispiel die Datenbankverbindungen und ähnliche Metainformationen nur noch an einer Stelle verwaltet werden müssen. Nachteilig ist, dass die Beans hierbei bestimmten Regeln folgen müssen:

- Bestimmte Namenskonventionen müssen eingehalten werden (die Speicherung versagt ohne Rückmeldung, wenn das nicht passiert ist).
- Jede Bean muss genau einer Tabelle in der Datenbank entsprechen.
- Ein Application Server muss eingerichtet und verwaltet werden.

Hibernate geht viele der Probleme an, welche die Benutzung von Entity Beans so schwierig machen:

- Es sind nahezu keine Änderungen am Quelltext nötig.
- Hibernate unterstützt die flexible Verteilung von POJOs in die Datenbanktabellen, es kann mehr als eine Klasse pro Tabelle oder mehr als eine Tabelle pro Klasse geben.
- Es ermöglicht einen einfachen Einsatz, zum Beispiel durch automatische Einrichtung der Datenbank; außerdem ist nicht zwingend ein Application Server notwendig.

3.2.3. EJB3 und Hibernate

Die nächste Version der EJB-Spezifikation basiert auf den Erfahrungen, die mit Hibernate und anderen Persistenzmanagern gemacht wurden. Sie befand sich zu Beginn der Projektgruppe noch im Beta-Stadium, wurde aber kurz danach offiziell veröffentlicht. Im Zuge der Entwicklung dieser Spezifikation hat der Hersteller des, von der Projektgruppe benutzten, *Application Servers* (siehe Abschnitt 3.6 auf Seite 25) Hibernate als Standard-Persistenzmanager für sein Produkt eingeführt.

3.2.4. Criteria

Hibernate unterstützt eine objektorientierte Art der Abfrage von persistenten Daten, die *Criteria*-API. Diese Technik stellt die gesamte Abfrage als eine Menge von Objekten dar, die nach und nach hinzugefügt werden können. Auf diese Weise kann eine Suche auch ohne manuelle Übersetzung in eine andere Abfragesprache zur Laufzeit dynamisch aufgebaut werden.

3.2.5. Verwendung in der Projektgruppe

Die Objekte, welche die Daten der Anwendung repräsentieren, werden komplett von Hibernate verwaltet. Auch die Suche über diese Datenbestände macht starken Gebrauch der Criteria-API, um die benötigten komplexen Datenbankanfragen zu erstellen.

3.3. Web Services

Unter dem Begriff *Web Services* werden üblicherweise all jene Dienste zusammengefasst, die über das Internet erreichbar sind und ihre Daten in Form von XML-Dokumenten austauschen. Für die darunter liegende Kommunikation werden Standardtechnologien eingesetzt. Dies ist normalerweise HTTP via TCP, möglich ist aber auch eine Übertragung mittels SMTP.

Dadurch, dass Web Services vollständig auf offenen Standards aufsetzen, kann eine sehr gute Plattform- und Programmiersprachenunabhängigkeit erreicht werden. So unterstützen wohl die meisten Systeme und Programmiersprachen HTTP und TCP, aber auch Parser für XML können wiederverwendet werden.

Im Folgenden werden die wichtigsten Standards für Web Services kurz angesprochen. Ferner werden die Anforderungen der Projektgruppe, welche die Verwendung von Web-Service-Technologien erfordern, erläutert.

3.3.1. Web-Service-Standards

Die zwei wohl wichtigsten Begriffe im Zusammenhang mit Web Services sind *WSDL* und *SOAP* (siehe [31] und [32]). Ersteres ist die *Web Service Description Language* und stellt eine auf XML basierende Beschreibungssprache für Web Services dar. Eine solche `.wsdl` Datei enthält alle Informationen, die ein potenzieller Client benötigt, um den dort beschriebenen Dienst auszuführen. Dazu gehören insbesondere der URL, unter dem der Service zu erreichen ist, der Name des Dienstes, Parameterdeklarationen, sowie Rückgabewert(e), wobei die Typen der letztgenannten plattformunabhänig beschrieben werden.

SOAP hingegen bezeichnet das Protokoll, mit dem die Nachrichten zwischen Web Service und Client ausgetauscht werden. Diese Nachrichten erfolgen in Form von speziellen XML-Dokumenten. Der Aufruf eines Web Service mittels SOAP enthält neben Rahmenelementen (*Envelope, Body, ...*) zum Beispiel die Parameter, die zur Ausführung an diesen Dienst übergeben werden müssen. In der Antwort werden dann die Rückgabewerte übertragen.

Ein zuvor mittels einer WSDL-Datei beschriebener Web Service kann also mit dem Protokoll SOAP aufgerufen werden.

3.3.2. Verwendung in der Projektgruppe

Web Services finden an zwei verschiedenen Stellen im Rahmen dieser Projektgruppe Verwendung. Zum einen wird die Suchfunktion als Web Service zur Verfügung gestellt, zum anderen erfolgt die Kommunikation zwischen Adminclient und -backend mittels Web Services.

Dadurch soll die Möglichkeit für alternative Frontends offengehalten werden, die nicht zwangsläufig ein HTML-Client unter der Verwendung von Servlets sein müssen. So wäre ein Java Swing Client mit verhältnismäßig geringem Aufwand zu erstellen, aber auch ganz andere Architekturen/Plattformen könnten realisiert werden, wie zum Beispiel C/GTK. Bei der Verwendung der Java-internen Möglichkeit RMI (Remote Method Invocation) wäre dies nicht möglich gewesen.

3.3.3. Auswahl eines Frameworks

Nach der Entscheidung, Web Services einzusetzen, galt es eine geeignete Implementierung auszumachen. Zunächst wurden die Vor- und Nachteile der beiden *Axis*-Versionslinien (siehe [2] und [3]) – von der *Apache Foundation* entwickelte freie Implementierungen des SOAP-Protokolls – untersucht. Das Ergebnis der Untersuchung war, dass Axis 2 aktuell noch unzureichend dokumentiert ist, sodass sich die Projektgruppe gegen die Verwendung entschieden hat.

Bei dem späteren Vergleich mit der *JSR-181*-Spezifikation für *Web Services Metadata* (siehe [13]) entschied sich die Gruppe, diese Variante statt Axis 1 einzusetzen. Sie ist Bestandteil der Java EE Spezifikationen und damit implementierungsunabhängig. Darüber hinaus beinhaltet der aktuell zur Verwendung vorgesehene Application Server *JBoss* (siehe Kapitel 3.6 auf Seite 25) bereits eine Implementierung.

Der große Vorteil dieser noch recht jungen[2] Spezifikation ist der, dass sie eine sehr einfache Möglichkeit bietet, Web Services zu erstellen. Die sonst sehr aufwendigen Arbeitsschritte (wie zum Beispiel das Erstellen der WSDL) werden hier größtenteils automatisiert. Hierfür muss lediglich die Annotation `@WebService` mit einigen wenigen Parametern an die Klasse geschrieben werden, sowie die Annotation `@WebMethod` an all jene Methoden in dieser Klasse, die als Web Service erreichbar sein sollen. In der `web.xml` wird die Klasse wie ein Servlet angemeldet. Beim Installieren der Web Application merkt dann der JSR-181-fähige Application Server, dass es sich bei der angemeldeten Klasse nicht um ein Servlet, sondern um einen Web Service handelt, woraufhin er die *WSDL*-Beschreibung automatisch generiert.

Die Klasse, die die Annotation `@WebService` trägt, sollte ein bestimmtes Interface implementieren, damit dieses auf Seite des Clients dazu verwendet werden kann, den Web Service zu repräsentieren. Wenn auch die Existenz eines solchen Interfaces nicht zwingend nötig ist, so stellt sie doch sicher, dass die am Server definierten Methoden zu denen passen, die der Client erwartet.

[2] *Final Release* der Spezifikation beim JCP am 27. Juni 2005

3.4. Servlets und JavaServer Pages

Die Kopplung zwischen *JSP*-Seiten und *Servlets* ist der Standardweg für die Implementierung der auf Java basierten Webanwendungen. Dabei übernimmt die JSP-Technologie die Darstellung und die Servlets beinhalten den Kontrollfluss der Anwendungslogik. Genau dieser klassische Ansatz wurde als Basis für die Verwirklichung der Weboberfläche von oCOBS (im Folgenden Webclient genannt) angewandt. Angesichts der Anforderungen der zu implementierten Anwendung bietet die reine Java EE Lösung eine Reihe von Vorteilen gegenüber innovativen Abstraktionsvorschlägen wie etwa *JavaServer Faces* und *Struts*. Die wichtigsten sind Effizienz, Flexibilität und Stabilität. Da der Webclient eine verhältnismäßig einfache Verlinkungsinfrastruktur besitzt und einzelne Webseiten wenige Elemente enthalten, ist die Projektgruppe zum Entschluss gekommen, im Webclient-Teil bei der JSP-Servlet-Alternative zu bleiben. Mehr zum Webclient gibt es im Kapitel 8 auf Seite 60.

3.4.1. JavaServer Pages

JSP ist eine Technologie, die Dynamik in statische Webseiten bringt. Anhand gewisser *XML*-Ausdrücke wird der dynamische Inhalt in ganz normale *HTML*-Seiten eingebettet. Intern wird eine JSP-Seite beim ersten Aufruf vom *Servlet Container* in ein Servlet umgewandelt. Demnach ist JSP eine Abstraktion der Servlets, deren Anwendung auf spezifische Bedürfnisse eingeschränkt ist.

Ein wichtiger Begriff im Zusammenhang mit JSP ist die *JavaBean*. Dabei geht es um eine Java-Klasse, die ausschließlich Getter und Setter implementiert. JavaBeans dienen als eine Art Vermittler bei der Übertragung von Daten zwischen JSP-Seiten und Servlets. JSP unterstützt eine Reihe von Möglichkeiten, um auf einfache Weise auf die JavaBeans zuzugreifen und ihre Inhalte zu verwalten.

Eine weitere mächtige Eigenschaft in JSP ist die Erweiterung durch *Custom Taglibs*. Der Entwickler ist somit nicht nur an die Standardfunktionen gebunden, sondern verfügt über die Möglichkeit, die „JSP-Sprache“ an seine Ansprüche anzupassen.

3.4.2. Servlets

Servlets stellen ganz normale Java-Klassen dar, welche die Clientanfragen entgegennehmen, bearbeiten und entsprechend darauf reagieren. Von Servlets aus gibt es die Möglichkeit, auf die *HTTP-GET-* und *HTTP-POST-Variablen* zuzugreifen, die vom Client an den Server übermittelt werden. Im Grunde genommen kann ein Servlet mühelos eine vollständige HTML-Seite erstellen und an den Client verschicken. So wird es auch in manchen Anwendungsfällen gemacht. Doch wenn es um die Implementierung einer anspruchsvollen Anwendung geht, erweist sich die Trennung zwischen Darstellung und Anwendungslogik als sinnvoll. Eben diese Erkenntnis führt

zum Zusammenspiel zwischen Servlets und JSP. Wie genau dieses Zusammenspiel aussieht, ob die Anfrage an eine JSP-Seite gerichtet ist, die ihrerseits ein Servlet importiert, oder umgekehrt, ist transparent. Essenziell ist das übersichtliche konzeptionelle Vorgehen.

3.5. JavaServer Faces

Die Komplexität heutiger Webanwendungen wie beispielsweise der DBLP stellt hohe Anforderungen an deren Entwickler. Techniken wie die oben vorgestellten *JavaServer Pages* und *Java Servlets* stellen mächtige Funktionen bereit, deren Kombination jedoch einen großen Aufwand erfordert.

Daher entschied sich die Projektgruppe, Teile der Anwendung mit Hilfe einer Framework-Technologie namens *JavaServer Faces* (kurz JSF) zu entwickeln, welche Entwickler bei deren Arbeit unterstützt und viele der sonst manuellen Abläufe automatisiert.

3.5.1. Möglichkeiten von JSF

Die klassische Verbindung von JSP mit Servlets gestaltet sich wie folgt: Führt der Benutzer der Seite eine Aktion aus, so werden die Daten an ein *Servlet* übermittelt. Dieses prüft die erhaltenen Informationen und führt Aktionen wie das Holen neuer Daten aus. Es übergibt diese an eine JSP-Seite, welche eine aktualisierte Ansicht für den Benutzer erzeugt.

Dieser eigentlich recht übersichtliche Ablauf erfordert einige Arbeit bei der Implementierung der Servlets. Sind die übertragenen Daten korrekt? Was geschieht im Falle von ungültigen Daten? Welche auf der Seite angegebenen Daten müssen wo gespeichert werden? Auf welche JSP-Seite soll als nächstes verwiesen werden?

Auch die JSP-Seiten sind hiervon betroffen: Wie werden übergebene Daten (beispielsweise `java.util.Date`) dargestellt?

JavaServer Faces schafft hier Abhilfe, indem es in der Entwicklung eine Abstraktionsebene höher ansetzt. Der Entwickler der JSP-Seiten generiert keinen HTML-Code mehr, sondern setzt seine Seite aus wiederverwertbaren JSF-Komponenten zusammen. Diese lassen sich im Verhalten und der Darstellung beliebig anpassen.

So ist es beispielsweise möglich, für eine Textbox festzulegen, welche Werte erlaubt sind. Die Validierung findet nicht mehr im Servlet statt, sondern wird bei Bedarf von JSF automatisch nach vorher festgelegten Regeln durchgeführt.

Weiterhin übernimmt JSF Aufgaben wie die der Navigation. In einer vorher definierten Datei ist angegeben, was geschehen soll, wenn auf einer bestimmten Seite ein bestimmtes Ereignis auftritt. Dadurch muss bei einer Änderung der Seitenstruktur nur eine Datei bearbeitet werden, die JSP-Seiten bleiben unberührt.

JSF ist dabei nicht auf eine Ausgabe in HTML beschränkt. Stattdessen ist es durch die Verwendung selbst erstellter Renderer möglich, etwa eine WAP-Version für Mobiltelefone zu entwickeln.

3.5.2. Nachteile

Da eine mit *JavaServer Faces* erzeugte Anwendung intern nur *JavaServer Pages* und *Servlets* benutzt, benötigt sie in der Regel nicht weniger Rechenleistung, als wenn die (effiziente) Implementierung manuell vorgenommen würde. Weiterhin ist mit einem stärkeren Netzwerkverkehr zu rechnen.

3.5.3. Verwendung in der Projektgruppe

Der öffentliche Webclient wird im Praxiseinsatz einen großen Benutzerandrang bewältigen müssen. Aus diesem Grund entschied sich die zuständige Gruppe gegen den Einsatz von JSF.

Im Gegensatz dazu wird die Administrationsoberfläche von deutlich weniger Personen benutzt, sodass das Argument der Rechenleistung hier weniger Gewicht hat. Auch ist es in dieser geschlossenen Personengruppe möglich, Annahmen über die verwendeten Browser oder aktivierten Techniken wie *JavaScript* zu treffen.

Die für die Administration zuständige Gruppe hat sich daher für die Verwendung von JSF entschieden. Hauptargumente für diese Wahl waren die Nutzung der vorgefertigten Validatoren und eine effiziente Arbeitsaufteilung unter den Mitgliedern.

3.6. JBoss

Um den reibungslosen Betrieb komplexer Webanwendungen zu gewährleisten, bedarf es Servern, auf denen diese Anwendungen bereitgestellt werden. Diese sogenannten *Application Server* entlasten Entwickler von Webanwendungen durch eine Vielzahl mächtiger Dienste und Funktionen, die über standardisierte Software-Schnittstellen angesprochen werden.

In der *3-Schichten-Architektur für Webanwendungen* befindet sich der Application Server auf der Geschäftslogikschicht und stellt somit das Bindeglied zwischen der Präsentationsschicht und der Persistenzschicht dar. Da die Präsentationsschicht und die Geschäftslogikschicht auf unterschiedlichen Serverinstanzen betrieben werden können, muss die Kommunikation dieser beiden Schichten durch zusätzliche Technologien gewährleistet werden. Häufige Verwendung finden hier *Remote Method Invocation* (kurz RMI) oder *Web Services* (siehe Abschnitt 3.3 auf Seite 21).

Ein Application Server besteht aus zwei Containern: Ein *Web Container* enthält die webbasierten GUI-Komponenten in Form von JSP-Seiten und Servlets (siehe

Abschnitt 3.4 auf Seite 23), während ein *Application Container* die Geschäftslogik der Anwendung verwaltet. Da die Geschäftlogik im Java EE 5 Kontext durch EJBs (siehe Abschnitt 3.1 auf Seite 17) implementiert wird, nennt man den Application Container, in dem die EJBs bereitgestellt werden, auch *EJB-Container*.

Mit dem *JBoss Application Server* (*JBoss*) lässt sich eine hohe Skalierbarkeit der Anwendung realisieren. Zudem bietet er alle Vorteile einer Open-Source-Software. Die für die Projektgruppe besonders relevanten Vorteile werden im nächsten Abschnitt kurz beschrieben.

3.6.1. Verwendung in der Projektgruppe

Der JBoss Application Server verdankt seine marktführende Position nicht nur dem Umstand, dass er als Open-Source-Software kostenlos ist. Er bietet zudem mächtige Clustering-Funktionalitäten, die eine gute Skalierbarkeit der Anwendungen gewährleisten. Von diesen Funktionalitäten hat die Projektgruppe zwar keinen Gebrauch gemacht, doch die Möglichkeit oCOBS zu einem späteren Zeitpunkt in einer geclusterten Umgebung zu betreiben bleibt einer nachfolgenden Projektgruppe vorbehalten. Zudem bietet der JBoss ab der Version 4 die Integration von *Hibernate* (Kapitel 3.2 auf Seite 19), welches bei oCOBS zur Abstraktion auf der Persistenzschicht (Kapitel 6 auf Seite 46) eingesetzt wird.

Des Weiteren möchte die Projektgruppe die Administrationsoberfläche (Abschnitt 11 auf Seite 90) vom Rest der Anwendung entkoppeln. Dazu soll oCOBS eine Web-Service-Schnittstelle anbieten, über welche die Kommunikation der Komponenten ermöglicht wird. Der JBoss bietet eine Unterstützung für Web Services durch seine sogenannte *JBoss WS* Engine. Erwähnenswert ist zudem das sogenannte *Hot Deployment* des JBoss. Diese Funktionalität ermöglicht es Anwendungen bereitzustellen, ohne dass der Server dazu neu initialisiert werden muss. Letzteres soll sich besonders in der Entwicklungs- und Testphase als Vorteil erweisen, da zeitaufwändige Neustarts des Servers entfallen.

Zuletzt bietet der JBoss mit dem *Apache Tomcat 5* einen modularen Web- und Servlet-Container Dienst an. Auch dieser erweist sich besonders in der Entwicklungsphase der Anwendung als vorteilhaft, da man speziell in dieser Phase die JSPs und Servlets (siehe Abschnitt 3.4 auf Seite 23) auf der Präsentationsschicht physikalisch nicht von den Komponenten der Geschäftslogikschicht entkoppeln möchte.

3.6.2. Weitere Vorteile

Der JBoss bietet über die gerade erwähnten, für die Projektgruppe besonders relevanten Funktionalitäten hinaus noch eine Vielzahl anderer Vorzüge, die an dieser Stelle kurz erläutert werden sollen:

- Der JBoss ist zu 100% J2EE 1.4 zertifiziert.

- Trotz der Vielzahl an administrativer Programmlogik, die ein Application Server bewältigen muss, sollte noch genug Rechenzeit für die eigentliche Anwendung zur Verfügung stehen. Hier haben Benchmark-Tests ergeben, dass die Performanz des JBoss wesentlich besser ist, als die anderer, kommerzieller Application Server.

- Durch standardisierte Schnittstellen und durch seine auf *Java Management Extensions* (JMX) aufbauende, komponentenbasierte Plugin-Architektur ermöglicht der JBoss eine leichte Administration vorhandener beziehungsweise ein leichtes Hinzufügen neuer Dienste.

- Dank einer großen Gemeinde aktiver Entwickler weltweit wird der JBoss ständig weiterentwickelt und verbessert.

Zu Beginn der Gruppenarbeit wurde die Version 4.0.4 des JBoss eingesetzt. Im Verlauf der Entwicklungsarbeiten erfolgte ein Umstieg auf die Version 4.0.5. In beiden Versionen wurden die EJB 3.0 Spezifikationen bereits umsetzt.

3.7. JavaABC

Das Java **A**pplication **B**uilding **C**enter (*JavaABC*) ist ein graphisches Tool zur grob granularen Modellierung von *Prozessen* beziehungsweise *Workflows*. Diese kann von einem Anwendungsexperten durchgeführt werden, der keine Programmiererfahrung hat. Die eigentliche Logik der einzelnen *SIBs*[3] des Graphen wird von einem IT-Spezialisten entwickelt. Diese Trennung von Koordination und Implementierung hat eine bessere Strukturierung des Projektes zur Folge und ist auch bei Aufgaben, an denen nur Programmierer arbeiten, hilfreich.

Das JavaABC bietet die Möglichkeit, über den gesamten Entwicklungsprozess hinweg ein *Modell* zu verwenden, das zu Beginn nur die Anforderungen spezifiziert und nach Fertigstellung des Projektes ein ausführbares Programm darstellt. Dadurch bleibt ein System – bei richtiger Anwendung – von der Spezifikation bis zur Auslieferung konsistent.

Weiterhin ist das *JavaABC Framework* sehr flexibel und anpassbar. Es bietet lediglich die Kernfunktionalität zur Modellierung und Ausführung von Graphen. Plugins erweitern die Funktionalität, so dass beispielsweise eine Schnittstelle zu einer Datenbank möglich ist. Die Modularisierung in einzelne Komponenten, den SIBs, entspricht dem *SOA-Ansatz*[4] [34] und ermöglicht eine einfache Integration von Software von Drittanbietern über vordefinierte Schnittstellen sowie die Einbindung von *Web Services*. Ferner kann ein Modell auf globale und lokale Regeln hin getestet

[3] Bezeichnung für einzelne Bausteine des Graphen; SIB - **S**ervice **I**ndependent Building **B**lock
[4] SOA - **S**ervice **O**riented **A**rchitecture

werden. Das JavaABC bietet die Möglichkeit, ein konsistentes, anpassbares, verifizierbares und ausführbares Modell eines Anwendungsablaufs auf einem sehr hohen Abstraktionsniveau zu gestalten.

3.7.1. Lightweight Process Coordination (LPC)

Der ausführbare SIB-Graph eines JavaABC-Modells enthält einzelne Komponenten (SIBs), die eine beliebig komplizierte Funktionalität beinhalten können. Es ist möglich, in einem SIB einen Web Service, einen Datenbankaufruf oder einen aufwändigen Algorithmus auszuführen. Der Graph modelliert lediglich die Koordination der Prozesse. Diese Form der Prozessmodellierung beziehungsweise -koordinierung nennt sich die *Lightweight Process Coordination*, die sehr nah an den Ansatz der dienstorientierten Architektur angelehnt ist. Das JavaABC Framework kann sehr leicht durch Plugins erweitert werden. Unter anderem ist es mit dem sogenannten Tracer-Plugin möglich, einen SIB-Graphen direkt auszuführen, sofern die einzelnen Komponenten des Graphen implementiert sind. Ein Trace-Graph lässt sich mit Hilfe des *Genesys*-Plugins (früher Codegenerator [27]) in ausführbaren Code konvertieren, der auch ohne die JavaABC-Anwendung ausführbar ist (es wird lediglich das Framework benötigt). Nähere Informationen zu der Erweiterbarkeit des Frameworks sind in [23] zu finden.

Die LPC hat folgende Eigenschaften [22]:

Einfachheit Der Prozess konzentriert sich auf Anwendungsexperten, die keine Programmiererfahrung benötigen. Die grundlegende Idee des Modellierungsprozesses ist in vergangenen Projekten neuen Teilnehmern innerhalb von weniger als einer Stunde erklärt worden.

Flexibilität Änderungen an dem Modell sind vorgesehen und behindern nicht andere Erweiterungen des Systems. Dies stimmt mit dem Prinzip der agilen Softwareentwicklung überein.

Anpassbarkeit Die LPC-Komponenten, die ein Modell bilden, können frei umbenannt oder umstrukturiert werden, um den Namenskonventionen und Bedürfnissen des Anwendungsexperten zu entsprechen.

Konsistenz Der Entwicklungsprozess mit dem JavaABC verwendet dasselbe Modell, von den ersten Schritten des Prototyping bis hin zur Fertigstellung.

Verifikation Mit Techniken wie dem *Local-* und *Modelchecking* wird der LPC-Nutzer unterstützt, während das Modell modifiziert wird. Die Idee ist, lokale und globale Bedingungen an ein Modell zu knüpfen, welche dieses während und vor allem am Ende des Entwicklungsprozesses erfüllen soll.

Dienstorientiert Bestehende Features oder Anwendungen können einfach in das LPC-Modell integriert werden, indem die Funktionalitäten in einer LPC-Komponente verpackt werden.

Ausführbarkeit Ein Modell kann verschiedene Ebenen von ausführbarem Code von Beginn der ersten Simulation an bis hin zur endgültigen Laufzeitimplementierung enthalten.

Universalität Das Framework nutzt die plattformunabhängige und objektorientierte Sprache Java als Basis des Systems und ist daher nicht beschränkt. Auch dem Zugriff auf andere Programmiersprachen und Architekturen sind kaum Grenzen gesetzt.

Die Vier-Schichten-Architektur

Das Entwicklungsparadigma des JavaABC beziehungsweise der LPC basiert auf der Erweiterung der klassischen *Drei-Schichten-Architektur*, um eine zusätzliche *Koordinationsschicht* zwischen der *Präsentationsschicht* und der *Komponentenschicht*[5]. Die SIB-Graphen koordinieren die Komponenten der Geschäftslogik.

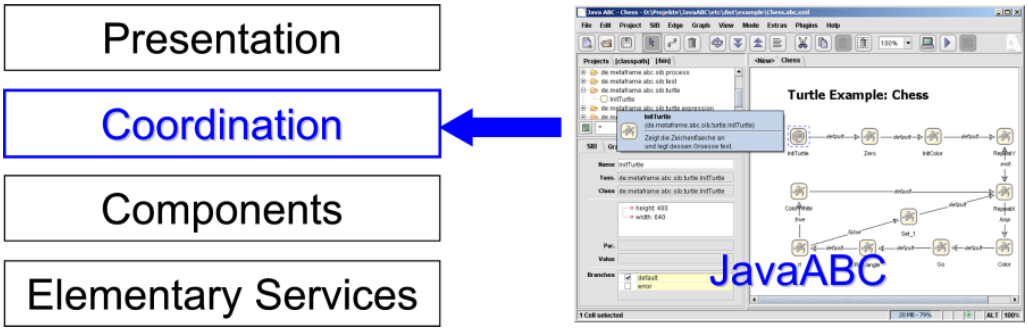


Abbildung 3.1.: Vier Schichten Architektur

3.7.2. SIB-Graph

In der JavaABC-Terminologie heißt ein Modell SIB-Graph. Ein SIB-Graph besteht aus Knoten[6] (siehe Kapitel 3.7.3 auf der nächsten Seite), benannten Kanten[7] (siehe [23]) sowie Modellparametern und -kanten. Die Parameter und Branches eines Modells sind globale SIB-Parameter und -Branches (siehe Kapitel 3.7.3 auf der nächsten Seite). Diese können in der Anwendung an einzelnen Knoten gesetzt werden. Sie

[5]Wird oft auch Geschäftslogikschicht genannt.
[6]Service Independent Building Blocks
[7]Branches

ähneln konzeptionell den globalen Variablen einer Klasseninstanz. Bei hierarchischen SIB-Graphen bilden diese Eigenschaften die Schnittstelle zu dem übergeordneten Graphen (siehe [23]).

3.7.3. Service Independent Building Blocks (SIBs)

Ein SIB entspricht einem Knoten in einem SIB-Graphen. SIBs sind wiederverwendbar und haben keine eigene Funktionalität, sondern greifen auf die Implementierung des SIB-Experten zurück. Die SIBs haben Parameter und ausgehende Kanten, sogenannte Branches, mit denen der Ablauf eines Graphen modelliert wird. Bei einem SIB handelt es sich um eine Java-Klasse. Die Funktionalitäten der verschiedenen Plugins, die ein SIB unterstützt, werden über die Implementierung des zugehörigen Interfaces realisiert. Im Idealfall sollen die SIB-Klassen den Umfang einer Karteikarte haben und dementsprechend nicht mehr als Eigenschaften und Verweise auf Methoden der eigentlichen Anwendung beinhalten. Damit wird die Funktionalität nicht in dem SIB angesiedelt, sondern an die Implementierung des SIB-Experten delegiert.

3.7.4. Architektur

Die JavaABC-Architektur ist in die drei Bereiche Implementierung, Modellierung sowie Speicherung/Versionierung eingeteilt. Die Implementierung wird von dem SIB-Experten vorgenommen. Dazu kann er seine favorisierte *Java IDE* verwenden. Dieser Bereich stellt der Modellierung die Komponenten in Form von SIBs zur Verfügung. Die Modellierung der Prozesse wird im JavaABC von einem Anwendungsexperten durchgeführt. Er sollte gute Kenntnisse über das Problem haben, benötigt jedoch keine Programmiererfahrung. Das Modell und die SIB-Klassen werden in einem *Versionskontrollsystem* abgelegt. Zusätzlich kann der Anwendungsexperte die SIBs seinen Bedürfnissen entsprechend umstrukturieren und umbenennen. Diese sogenannten *Taxonomie-Daten* werden ebenfalls im Versionskontrollsystem gespeichert und versioniert.

SIB-Klassen

SIB-Klassen können unterschiedliche Ebenen von ausführbarem Code enthalten. Der Anwendungsexperte kann während des Modellierungsprozesses zum Beispiel auf eine Komponente stoßen, die er benötigt. Der SIBCreator-Plugin ermöglicht ihm, eine Dummy-SIB-Klasse zu erstellen, die später implementiert wird, damit er sofort weiter modellieren kann. Im gleichen Modell kann auch eine SIB-Klasse vorhanden sein, die bereits ihre Funktionalität zur Verfügung stellt.

Die projektspezifischen SIB-Klassen werden vom SIB-Experten in das Projektverzeichnis gelegt, damit sie bei der Modellierung und der späteren Ausführung zur

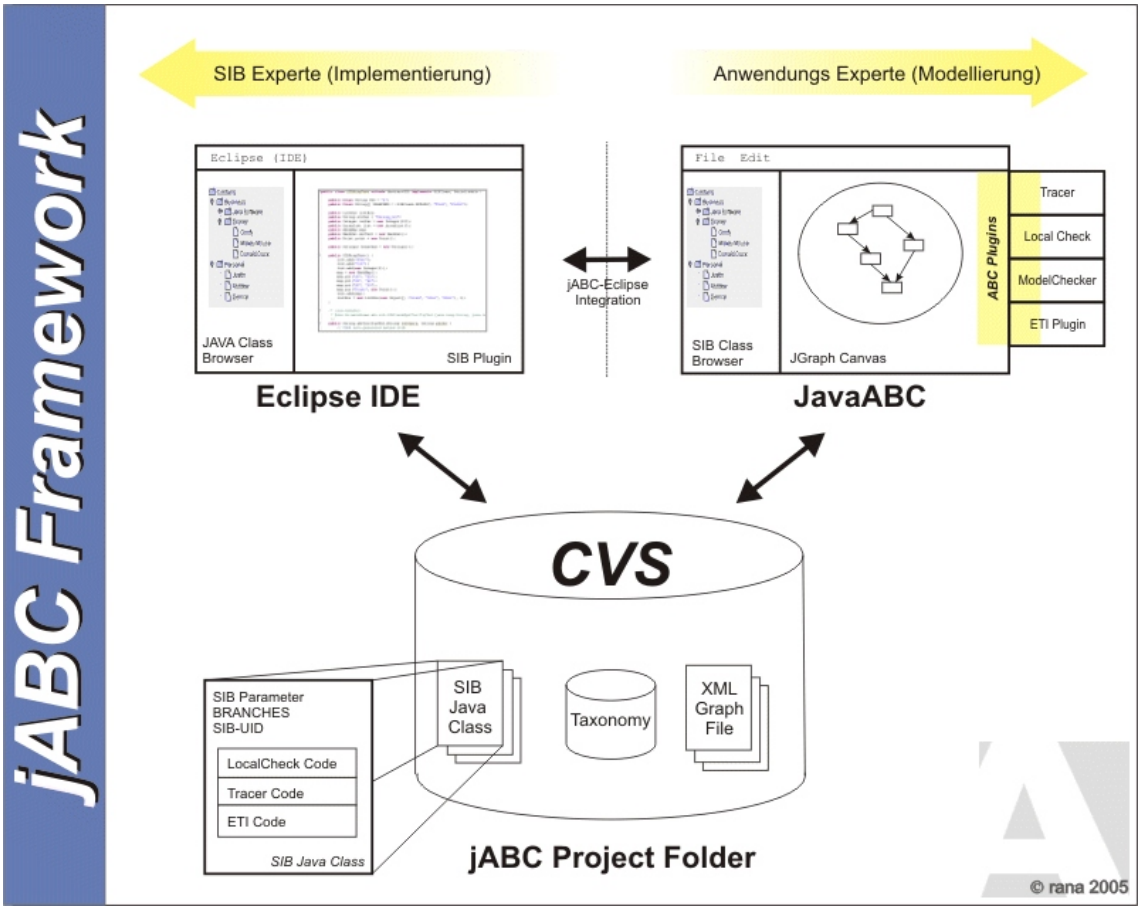


Abbildung 3.2.: Architektur des JavaABC

Verfügung stehen.

Taxonomie

Die Taxonomien im JavaABC übernehmen die Funktion, dem Anwendungsexperten eine vertraute Umgebung zu ermöglichen, ohne den Programmierer einzuschränken. Die Struktur und Benennung der SIBs kann mit dem Taxonomieeditor komplett geändert werden, ohne die Verzeichnisstruktur zu verändern. Es existiert genau eine Taxonomie pro SIB-Pfad.

XML-Graph-Datei

Die XML Graph Dateien enthalten alle Informationen über die SIBs eines Modells, ihre Parameter und die Kanten zwischen den SIBs sowie deren Beschriftung.

3.7.5. Plugins

Das Konzept des JavaABCs ermöglicht eine starke Trennung zwischen der Grundfunktionalität der Modellierung und den Zusatzfunktionen (wie Graphenausführung, Local- und Modelchecking, Quelltextgenerierung, Datenbankzugriff und viele weitere). Ein Plugin muss im JavaABC lediglich die Plugin-Schnittstelle implementieren. Über den Zugriff auf das Framework kann sich ein Plugin nahtlos in die Anwendung einbinden.

3.7.6. Verwendung in der Projektgruppe

Das JavaABC findet in mehreren Bereichen Anwendung:

- Im Webclient wird die Verlinkung zwischen den einzelnen Webseiten mit dem JavaABC modelliert. Das jEEWAB-Plugin (siehe [23]) ermöglicht die Modellierung von SIB-Graphen, welche die statische Struktur einer Webanwendung widerspiegeln.
- Der Kontrollfluss des Importvorgangs ist mit dem JavaABC modelliert worden. Für das resultierende Modell wurden grundlegende Korrektheitsaspekte wie etwa eine ordnungsgemäße Transaktionsverwaltung mit Hilfe des FormulaBuilder-Plugins [26] spezifiziert und schließlich mit dem GEAR-Plugin [18] durch Model-Checking verifiziert. Das Genesys-Plugin erzeugte aus dem Modell den ausführbaren Java-Code, der dann in die Anwendung integriert wurde.
- Für das Datenmodell wird das JavaABC verwendet, um das von Hibernate erstellte Datenbankschema mit Hilfe der Reverse-Engineering-Fähigkeit des DBSchema-Plugins [7] in einem Modell darzustellen und beispielsweise für den Suchalgorithmus nachzuvollziehen.
- Weiterhin wurde der Export (siehe „Export“ auf Seite 65) des Merkzettels mit Hilfe des JavaABC modelliert.

3.8. Model-Checking

Das automatisierte Testen von Produkten gehört seit geraumer Zeit zum Alltag in der Software-Entwicklung und wird speziell im Kontext der Java-Programmierung durch Werkzeuge wie *JUnit* [14] und *TestNG* [28] unterstützt. Mit zunehmender Größe und Komplexität einer Komponente steigt aber die Schwierigkeit, geeignete Testszenarien zu identifizieren, um alle möglichen Programmabläufe abzudecken. Hier bietet Model-Checking eine Möglichkeit zur automatisierten und vollständigen Verifikation.

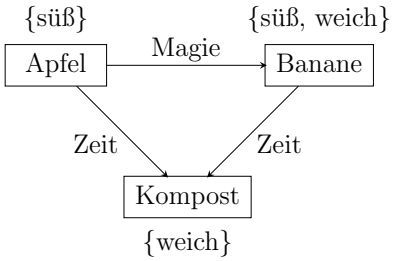
Allgemein bezeichnet Model-Checking das Entscheidungsproblem, ob ein gegebenes Modell $\mathcal{M}$ eine Eigenschaft ϕ erfüllt, ob also $\mathcal{M} \models \phi$ gilt. Nachfolgend soll dieses Konzept durch exemplarische Betrachtungen in seinen Grundzügen dargestellt werden. Für eine tiefergehende Einführung in die Thematik des Model-Checking sei an dieser Stelle etwa auf [21] verwiesen.

3.8.1. Modelle

Im Kontext des Model-Checking versteht man unter einem Modell eine konkrete Instanz eines Transitionssystems, also ein automatenähnliches Gebilde mit Zuständen und Übergängen zwischen diesen Zuständen, das die Dynamik einer zu untersuchenden Komponente – oder einer passenden Abstraktion davon – darstellt.

Anschaulich können solche Transitionssysteme durch gerichtete Graphen dargestellt werden, wobei Knoten die Zustände und Kanten die möglichen Zustandsübergänge repräsentieren. Allerdings gibt ein klassischer Graph zunächst einmal nur die prinzipielle Erreichbarkeit von Zuständen wieder. Um auch Aussagen darüber treffen zu können, welche Eigenschaften die jeweiligen Zustände haben und unter welchen Bedingungen Zustandsübergänge möglich sind, werden Knoten und Kanten mit Zusatzinformationen beschriftet.

Bei einem *Kripke-Transitionssystem* – um nur eine Form der Modellierung zu nennen – werden die Zustände mit einer Menge von atomaren Aussagen und die Zustandsübergänge jeweils mit einer Aktion beschriftet, wie in Abbildung 3.3 veranschaulicht.



$$
\begin{aligned}
&\mathcal{KTS} = (\mathit{States}, \mathit{Actions}, \longrightarrow, \mathit{Interpretation}) \text{ mit} \\
&\mathit{States} \quad = \quad \{\text{Apfel}, \text{Banane}, \text{Kompost}\} \\
&\mathit{Actions} \quad = \quad \{\text{Magie}, \text{Zeit}\} \\
&\longrightarrow \quad = \quad \{(\text{Apfel}, \text{Magie}, \text{Banane}), \\
&\qquad\qquad\quad (\text{Apfel}, \text{Zeit}, \text{Kompost}), \\
&\qquad\qquad\quad (\text{Banane}, \text{Zeit}, \text{Kompost})\} \\
&\mathit{Interpretation} \quad = \quad \{\text{Apfel} \mapsto \{\text{süß}\}, \\
&\qquad\qquad\quad \text{Kompost} \mapsto \{\text{weich}\}, \\
&\qquad\qquad\quad \text{Banane} \mapsto \{\text{süß}, \text{weich}\}\} \\
&\text{über den Aussagen } AP = \{\text{süß}, \text{weich}\}
\end{aligned}
$$

Abbildung 3.3.: Beispiel eines Kripke-Transitionssystems

Betrachtet man in einem Transitionssystem nicht nur einen Zustandsübergang sondern eine Kette aufeinanderfolgender Zustandsübergänge, so spricht man von einem Pfad. Pragmatisch beschreibt ein Pfad eine mögliche Berechnung der modellierten Komponente. Formal lässt sich solch ein Pfad etwa als Sequenz der durchlaufenen Zustände darstellen und weiter formal analysieren, um so Rückschlüsse auf die repräsentierte Berechnung zu ziehen.

3.8.2. Temporallogiken

Auf die formale Darstellung des zu verifizierenden Systems folgt die Formalisierung der zu verifizierenden Eigenschaften selbst. Unter einer Eigenschaft wird in diesem Kontext eine Aussage verstanden, die für die einzelnen Zustände des betrachteten Transitionssystems entweder erfüllt ist oder eben nicht. Das formale Werkzeug, um solche Eigenschaften darzustellen, sind boole'sche Logiken.

Um nun Aussagen über das dynamische Verhalten eines Systems treffen zu können, bedarf es entsprechend solcher Logiken, die den zeitlichen Aspekt berücksichtigen, sogenannter Temporallogiken. Typische Vertreter dieser Gruppe sind CTL, LTL, CTL* oder auch der modale μ-Kalkül.

Die *Computation Tree Logic* (CTL) als Beispiel erweitert die klassische Aussagenlogik zunächst um zwei Operatoren zur All- beziehungsweise Existenzquantifizierung über der Menge d von einem Zustand ausgehenden Pfade. Über die Zustände so quantifizierter Pf lassen sich dann weitere Aussagen treffen. Einen Eindruck solcher Aussagen vermittelt Abbildung 3.4, die jeweils einen Berechnungsbaum zeigt, dessen Ausgangszustand an der Wurzel die angegebene Formel erfüllt.

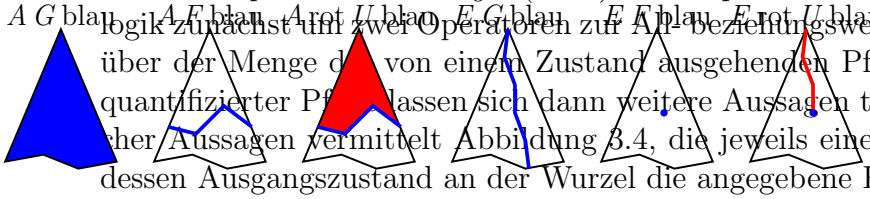


Abbildung 3.4.: Illustration der Semantik von CTL

3.8.3. Verwendung in der Projektgruppe

Wie schon am Ende von Abschnitt 3.7 auf Seite 27 dargestellt, wurde der Kontrollfluss des Importvorgangs auf grobgranularer Ebene mit dem JavaABC als SIB-Graph modelliert. Dieser SIB-Graph lässt sich auf naheliegende Weise in ein Kripke-Transitionssystem überführen, indem die SIBs als Zustände und die Branches als aktionsbeschriftete Zustandsübergänge interpretiert werden. Einzig die atomaren Aussagen, die für einzelne SIBs gelten, müssen zusätzlich erfasst werden.

Damit ist der SIB-Graph unmittelbar als Eingabe für das Model-Checking verwendbar, so dass sich etwa eine ordnungsgemäße Transaktionsverwaltung auf allen Berechnungspfaden des Imports verifizieren lässt.

Das eigentliche Model-Checking wird im JavaABC durch das GEAR-Plugin bewerkstelligt, welches Formeln nach CTL- oder μ-Kalkül-Syntax auswerten kann. Die Spezifikation der zu verifizierenden Eigenschaften erfolgte wiederum im JavaABC mittels des FormulaBuilder-Plugins, das aus dem azyklischen Syntaxgraph einer Formel für eine gewählte Zielsyntax die entsprechende Formel generiert.

Teil II.

Entwicklung

4. Konzept

Die DBLP in ihrer ursprünglichen Form stellt eine nicht kommerzielle Bibliographie dar, welche zu Gunsten einer hohen redaktionellen Datenqualität und -kontrolle bewusst auf die automatisierte Datenerfassung und -pflege durch ein *Content-Management-System* verzichtet. Hieraus ergeben sich leider einige Nachteile, wie zum Beispiel eine ineffiziente Datenpflege und Datenhaltung.

Für die Projektgruppe ist es eine der großen Herausforderungen, diese Nachteile in oCOBS, der Neukonzeptionierung der DBLP, aufzuheben und dennoch das hohe Maß an Datenqualität beizubehalten. Hieraus resultieren besondere Anforderungen an das Datenmodell (siehe Abschnitt 5 auf Seite 38) und den Datenimport (siehe Abschnitt 7 auf Seite 49).

Die zu implementierende Anwendung wird in drei große logische Bereiche eingeteilt, nämlich in den *Webclient*, eine *Administrationsoberfläche* und die Konzeptionierung eines *Datenmodells*, welches durch eine komplexe *Importroutine* initialisiert werden muss. Jeder dieser Bereiche erfüllt ein bestimmtes Spektrum der Aufgaben und leistet seinen Beitrag zu der Gesamtfunktionalität des Systems.

Die Aufgabe des Webclients ist es, die Interaktion mit dem Benutzer und die von dem System angebotenen Bibliographieverzeichnisdienste zur Verfügung zu stellen. Diese Dienste unterstützen das Auffinden der Publikationen und der Autoren im Verzeichnis und sind für alle Nutzer zugänglich (siehe Abschnitt 8 auf Seite 60).

Der Zugriff über die Administrationsoberfläche ist im Gegensatz dazu nur einem eingeschränkten Kreis der Benutzer möglich. Dabei geht es um die Registrierung neuer Publikationen und Autoren, die Überwachung des Systems, die Benutzerverwaltung und so weiter. Im Laufe der Konzeptionierung hat sich eine weitere Aufteilung der Administrationskomponente als sinnvoll herausgestellt. Die Idee dahinter war dabei, eine Trennlinie zwischen dem aufwändigen Webinterface und der tatsächlichen Administrationslogik zu ziehen. Eine detaillierte Beschreibung der Administrationsoberfläche ist in Abschnitt 11 auf Seite 90 zu finden.

Besondere Aufmerksamkeit wurde den Themen Datenmodell und Datenimport gewidmet. Das Datenmodell muss zum Schutz des Datenbestandes ein Benutzerrollen- und Rechtemanagement unterstützen. Ferner muss es für die einzelnen Publikationen Zustände verwalten können. Über diese Zustände soll es für mit administrativen Rechten ausgestatteten Benutzern später über die Administra-

tionsoberfläche (siehe Abschnitt 11 auf Seite 90) möglich sein, eine Beurteilung der Publikationen vorzunehmen, bevor diese zur Veröffentlichung freigegeben werden. Zudem sollen die Administratoren über ein spezielles Task-System benachrichtigt werden, falls bestimmte Änderungen im Datenbestand ihre Aufmerksamkeit erfordern. Um Änderungen im Datenbestand und deren Verursacher nachverfolgen zu können, bedarf es außerdem eines umfassenden Logging-Systems, welches alle datenverändernden Vorgänge protokolliert und diese persistent speichert. Für eine detaillierte Beschreibung des Datenmodells siehe Abschnitt 5 auf der nächsten Seite.

Für den Datenimport gelten ähnliche Anforderungen wie für das Datenmodell. Auch hier haben die Richtigkeit und die Wahrung der Konsistenz der Daten speziell beim inkrementellen Import oberste Priorität. Unter keinen Umständen darf der Importalgorithmus zulassen, dass eine Änderung im Format der zu importierenden Daten zu einem inkonsistenten Zustand in der Anwendung führt. Speziell beim inkrementellen Import müssen die bereits vorhandenen Daten mit den zu importierenden verglichen und Aktualisierungen erkannt werden. Entstehen Versionskonflikte zwischen diesen beiden Datenbeständen, so müssen die Administratoren über geeignete Tasks mit der Behebung dieser betraut werden. Für eine detaillierte Beschreibung des Datenimports siehe Abschnitt 7 auf Seite 49.

Alle eben im Zusammenhang mit der Administrationsoberfläche, dem Datenmodell sowie dem Import erwähnten Vorkehrungen und Mechanismen zielen vor allem darauf ab die aus der DBLP gewohnte hohe Datenqualität und redaktionelle Kontrolle auch in oCOBS aufrecht zu erhalten.

Die durchgeführte Partitionierung verleiht dem Entwicklungsprozess ein großes Maß an Flexibilität. Vorteilhaft ist vor allem die Möglichkeit der Nutzung verschiedener Technologien, die je nach Aufgabenbereich eingesetzt werden können. Das lebhafte Beispiel dazu ist die Verwendung der JSP- und Servlet-Technologie bei der Implementierung des Webclients, bei dem die Hauptanforderung Effizienz ist. Dagegen kommt bei der Administrationskomponente die JSF-Technologie zum Einsatz, da hier das Effizienzkriterium aufgrund der eingeschränkten Benutzerzahl eher sekundär ist.

5. Datenmodell

Im Folgenden werden die Entitäten des Problembereichs und ihre Beziehungen untereinander näher beschrieben, auf denen dann die diversen Dienste der Anwendung aufsetzen.

Für die Persistenz dieser Daten kommt ein relationales Datenbanksystem zum Einsatz, wobei die Transformation zwischen objektorientierter und relationaler Darstellung nach einmaliger Konfiguration durch *Hibernate* (siehe Abschnitt 3.2 auf Seite 19) vollkommen automatisch geschieht. Details der relationalen Modellierung wie Primärschlüssel oder Versionszähler werden an dieser Stelle nicht behandelt und können stattdessen der projektbegleitenden technischen Dokumentation entnommen werden.

5.1. Bibliographie

Als Vorbereitung zur Modellierung der Entitäten aus dem Bereich Literaturdaten wurden zunächst die bestehenden Datenbestände der DBLP analysiert. Wie bereits im einleitenden Teil zur DBLP geschildert (siehe Abschnitt 2.4 auf Seite 10), sind hierbei prinzipiell die bibliographischen Datensätze sowie der bibliographische Hypertext zu unterscheiden.

Die Gesamtheit der bibliographischen Datensätze wird regelmäßig in einer validierbaren XML-Datei öffentlich zur Verfügung gestellt. Der folgende Ausschnitt aus der zugehörigen DTD-Datei veranschaulicht den prinzipiellen Aufbau der bibliographischen Datensätze:

```
<!ELEMENT dblp
  (article|inproceedings|proceedings|book|
   incollection|phdthesis|mastersthesis|www)*>

<!ELEMENT inproceedings
  (author|editor|title|booktitle|pages|year|
   address|journal|volume|number|month|url|ee|
   cdrom|cite|publisher|note|crossref|isbn|
   series|school|chapter)*>
<!ATTLIST inproceedings key CDATA #REQUIRED>
```

Derzeit werden acht Typen von Publikationen unterschieden, denen jeweils fast zwei Dutzend verschiedene Datenfelder in beliebiger Anzahl zugeordnet werden können. Damit liegen bereits auf syntaktischer Ebene kaum Einschränkungen an die Inhalte der bibliographischen Datensätze vor. Auch ein näherer Blick auf die Inhalte selbst bestätigt, dass das Vorhandensein der Datenfelder von Publikation zu Publikation stark variiert, selbst bei Publikationen des gleichen Typs. Diese Unregelmäßigkeiten im Aufbau der bibliographischen Datensätze galt es bei der Modellierung zu berücksichtigen. Detaillierte Angaben zur Verteilung der Datenfelder können etwa bei [29, Anhang A.2] nachgelesen werden.

Ein nicht unmittelbar aus den bibliographischen Datensätzen ersichtlicher Aspekt ist die Problematik bei der Identifikation von Autoren und Editoren mittels ihrer Namen. Zum einen können zwei unterschiedliche Personen ein und denselben Namen besitzen. Zum anderen kann sich der Name einer Person im Laufe ihres Lebens ändern, so dass von dieser Person mitverfasste Publikationen unterschiedliche Namen im Datenfeld `author` aufweisen können. Für eine korrekte Erstellung der Autorenseiten oder des Koautorgraphen musste diese Problematik entsprechend behandelt werden.

Der bibliographische Hypertext wird durch die Betreiber der DBLP gleichfalls in einer validierbaren XML-Datei öffentlich verfügbar gemacht. Die dort enthaltenen Hypertextdokumente zur Generierung der ansonsten statischen HTML-Seiten der DBLP stellen im Wesentlichen eine verzeichnisartige Navigationsstruktur bereit. Auch diese Navigationsstruktur galt es neben den reinen bibliographischen Datensätzen geeignet zu berücksichtigen.

Obige Beobachtungen führten zusammen mit der Forderung nach einfacher Erweiterbarkeit zu den in Abbildung 5.1 auf der nächsten Seite dargestellten Entitäten und ihrer Beziehungen untereinander. Die nachfolgenden Erläuterungen sollen den Entwurf der Projektgruppe weiter begründen.

Ein grundlegendes Konzept ist die Modellierung der meisten Datenfelder durch die Klasse `Attribute` beziehungsweise ihrer konkreten Unterklassen. Die Gesamtmenge der Attribute ist dabei nicht zur Entwurfszeit fest definiert, sondern kann auch im späteren Betrieb der Anwendung geändert werden, womit die gewünschte Flexibilität erreicht wird.

Jedes Attribut hat einen Namen, der es bezogen auf eine der Unterklassen eindeutig identifiziert, sowie eine kurze Beschreibung und einen Typbezeichner. Die Menge der unterstützten Typbezeichner ist fest vorgegeben und deckt im Wesentlichen die üblichen primitiven Datentypen ab. Neben der Möglichkeit zur strengeren Validation der Benutzereingaben bietet dieser Typbezeichner der Präsentationsschicht einen Hinweis auf die angemessene Darstellung von Attributwerten. Darüber hinaus verfügt jedes Attribut über eine boole'sche Eigenschaft zur Festlegung, ob die mit diesem Attribut verbundenen Informationen öffentlich angezeigt werden sollen oder ob die betreffenden Angaben lediglich die administrativen Vorgänge zur Wartung

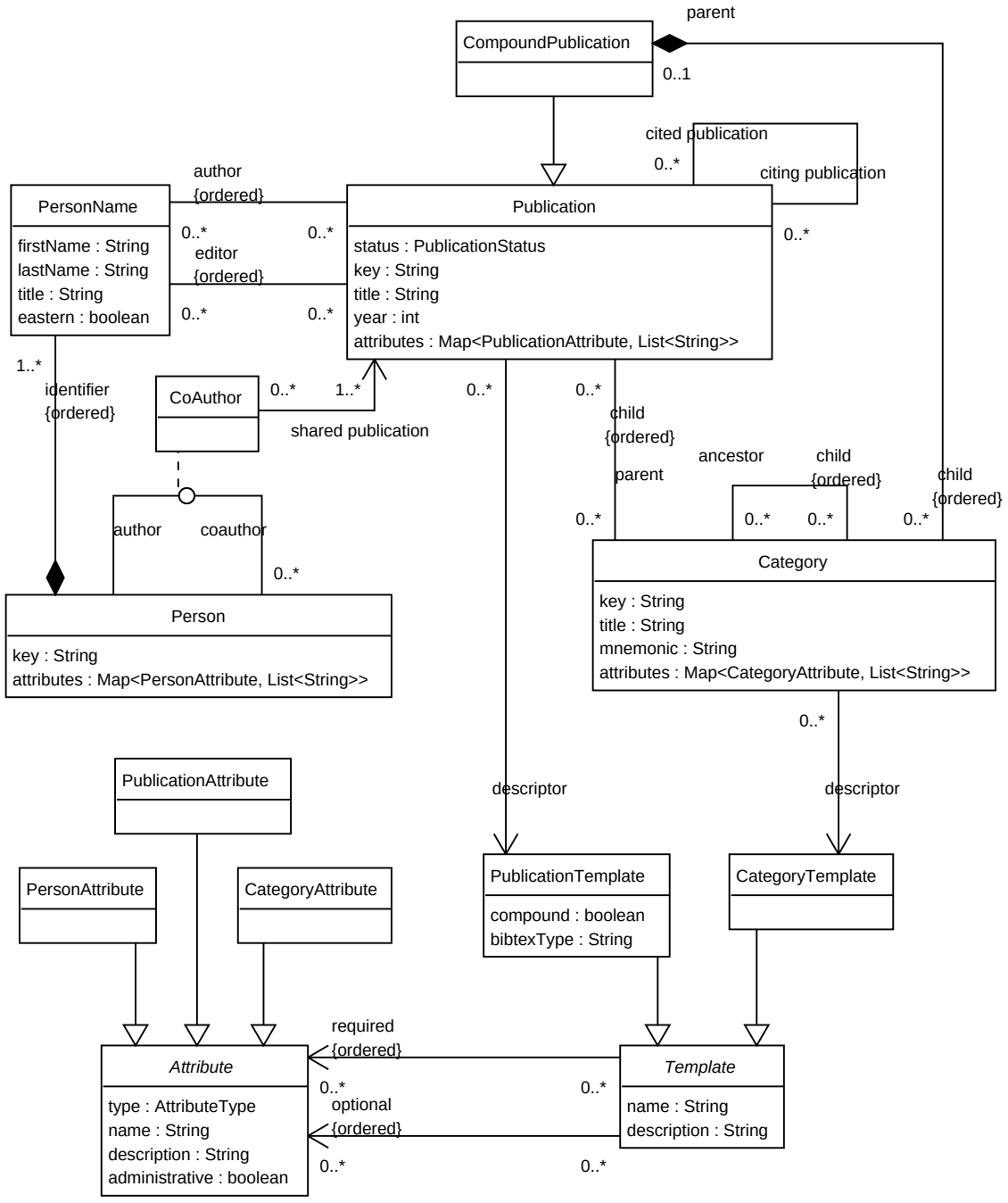


Abbildung 5.1.: UML-Klassendiagramm der Literaturdaten

der Bibliographie unterstützen.

Verwendung finden diese Attribute als Schlüssel für ein Wörterbuch in den Klassen `Person`, `Category` und `Publication`. Das betreffende Wörterbuch nimmt die meisten der Datenfelder für die jeweiligen Entitäten auf, indem zu einem Attribut eine Liste seiner Werte abgelegt wird. Diese Art der Speicherung wird insbesondere der starken Variation im Vorkommen einzelner Datenfelder gerecht und vermeidet unnötige Leerfelder. Die Verwendung des Attributs selbst als Schlüssel anstatt seines Namens erlaubt eine nachträgliche Umbenennung, ohne dass bestehende Wörterbücher geändert werden müssen.

Die Werte der Attribute selbst werden ungeachtet ihres zugehörigen Typbezeichners intern stets in Form von Zeichenketten gespeichert. Zur Serialisierung der Attributwerte innerhalb einer Datenbank musste ein allgemein verwendbarer Datentyp gefunden werden, der insbesondere auch ein einfaches Durchsuchen und Filtern des Datenbestandes anhand von Attributwerten gestattet.

Ein ähnliches Konzept wie bei den Attributen findet sich bei der Klasse `Template` und ihren Unterklassen. Diese Entitäten dienen als Typdeskriptoren für Objekte der Klassen `Publication` und `Category` und sind ebenso wie die Attribute nicht zur Entwurfszeit festgelegt, sondern können im späteren Betrieb geändert werden.

Jeder Typdeskriptor hat einen eindeutigen Namen und eine kurze Beschreibung. Weiterhin können optionale und erforderliche Attribute mit einem Typdeskriptor verknüpft werden, um beim Anlegen neuer Entitäten basierend auf dem Typdeskriptor ein entsprechendes Formular anzeigen und validieren zu können.

Obgleich das Namensfeld in den Klassen `Attribute` und `Template` das identifizierende Merkmal der betreffenden Entitäten ist, sollte auch diese Angabe durch die Administratoren der Anwendung frei definierbar sein. Damit ist die inhaltliche Bedeutung eines Attributs oder Typdeskriptors prinzipiell unabhängig von dem Namen des Objekts. An vielen Stellen der Anwendung ist aber eine Fallunterscheidung basierend auf der Bedeutung eines Objektes notwendig, um bei Anzeige, Import oder Export von bibliographischen Daten sachgemäß handeln zu können. Zum Schließen der durch die Flexibilität der Namensgebung bedingten Diskrepanz zwischen Syntax und Semantik kommt daher eine konfigurierbare Namenstabelle zum Einsatz. Diese Tabelle verknüpft den frei gewählten tatsächlichen Namen eines Attributs oder Typdeskriptors jeweils mit einem festen Standardnamen, über welchen anwendungsintern die Funktion und Bedeutung des Objekts ermittelt wird.

Die Klasse `Person` repräsentiert gleichermaßen die Autoren und Editoren von Publikationen, allerdings gibt es keine direkte Beziehung zur Klasse `Publication`. Vielmehr werden einer Person nur ihre Namen in Form von Objekten der Klasse `PersonName` zugeordnet. Erst die Namen sind dann den einzelnen Publikationen zugeordnet. Diese Modellierung erlaubt den sachgemäßen Umgang mit Namen und sichert etwa auch bei Export und Anzeige von bibliographischen Informationen die Angabe des historisch korrekten Personennamens unabhängig vom aktuellen

Namen der betreffenden Person.

Bei den Publikationen werden zwei Klassen unterschieden. Die einfachen Publikationen wie Fachartikel werden durch Objekte der Klasse `Publication` repräsentiert. Publikationsformen wie Sammel- oder Konferenzbände werden dagegen als zusammengesetzte Publikationen verstanden, in dem Sinne, dass sie einfache Publikationen enthalten, und werden daher durch die Unterklasse `CompoundPublication` modelliert.

Allen Publikationen ist der obligatorische Titel sowie das Erscheinungsjahr gemein. Zwecks Unterstützung inkrementeller Importe der bestehenden DBLP-Daten wird außerdem der eindeutige DBLP-Bezeichner gespeichert. Ein weiteres Feld speichert für administrative Vorgänge den Status einer Publikation, wobei die folgenden vier Zustände unterschieden werden:

- `INCOMPLETE`. Dieser Status markiert eine Publikation, die gerade neu erfasst wird und deren bibliographische Informationen noch nicht vollständig sind.
- `PROPOSED`. Hiermit wird eine Publikation gekennzeichnet, deren Neuerfassung oder Überarbeitung abgeschlossen ist und die nun in einer abschließenden Endkontrolle geprüft werden soll.
- `ONLINE`. Dies kennzeichnet Publikationen, die Teil des öffentlich sichtbaren Datenbestandes sind.
- `REJECTED`. In diesen Zustand werden Publikationen versetzt, die bis auf weiteres vor der Öffentlichkeit verborgen werden sollen.

Die verzeichnisartige Navigationsstruktur aus dem bibliographischen Hypertext wird schließlich durch je ein Objekt der Klasse `Category` pro Knoten im Verzeichnis verkörpert. Der azyklische Verweisgraph zwischen den Knoten spiegelt sich in einer Vorfahr-Kind-Beziehung wider.

Neben dem obligatorischen Titel wird für jeden Knoten aus der Navigationsstruktur aus bereits bekannten Gründen sein ursprünglicher DBLP-Bezeichner gespeichert. Ein benutzerdefinierbares mnemonisches Kürzel soll bei administrativen Vorgängen den Zugriff auf bestimmte Knoten beschleunigen.

5.2. Benutzerverwaltung

Zur präzisen Kontrolle der administrativen Vorgänge in der Anwendung werden den registrierten Mitarbeitern über Rollen feingranulare Zugriffsrechte zugewiesen. Die effektiven Zugriffsrechte eines Benutzers ergeben sich dabei aus der Addition aller Zugriffsrechte der ihm zugewiesenen Rollen.

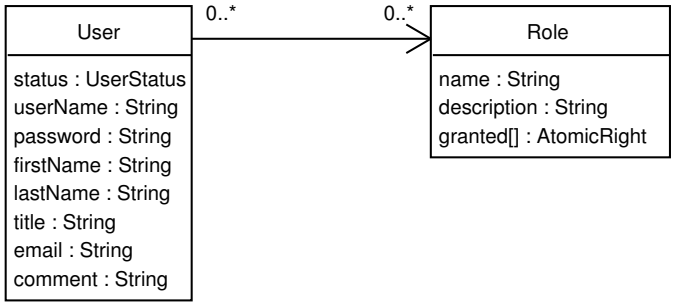


Abbildung 5.2.: UML-Klassendiagramm der Benutzerverwaltung

Das gesamte Modell der Benutzerverwaltung ist in Abbildung 5.2 dargestellt. Lediglich die Aufzählung `AtomicRight` der definierten Zugriffsrechte ist hier nicht näher beschrieben und kann stattdessen der Dokumentation des Quellcodes entnommen werden.

Um registrierte Benutzer vorübergehend oder endgültig vom System auszuschließen, kann deren zweiwertiges Statusfeld von `ACTIVE` auf `INACTIVE` gesetzt werden. Ein vollständiges Löschen der Entitäten vom Typ `User` ist im Hinblick auf eine lückenlose Protokollierung der Manipulationen des Datenbestandes nicht gewünscht.

5.3. Protokollierung

Zwecks detaillierter Nachverfolgung der Aktionen im Bereich Administration wird unabhängig von eventuell vorhandenen Protokolldaten eines Webservers eine eigenständige Protokollierung sämtlicher Manipulationen durchgeführt. Um die so erzeugten Protokolldaten auch maschinell schnell und effizient auswerten zu können – etwa zur Gewinnung von Statistiken – werden die einzelnen Ereignisse nicht in Form von einfachen Textmeldungen sondern durch dedizierte Klassen erfasst, welche in Abbildung 5.3 auf der nächsten Seite illustriert sind.

Ausgehend von einer Grundmenge allgemeiner Datenfelder für jedes zu protokollierende Ereignis wie dessen auslösender Benutzer, dem Zeitstempel und einer Beschreibung der geänderten Werte gibt es spezielle Unterklassen entsprechend den von der Benutzerverwaltung unterschiedenen atomaren Operationen. Die an einigen Stellen konzeptionell unnötigen abstrakten Basisklassen dienen dabei lediglich einer einfacheren und portableren Abbildung auf eine relationale Datenbank für die spätere Persistenz.

Die Motivation hinter dieser ausgeprägten Spezialisierung innerhalb der abgebildeten Klassenhierarchie ist die Bereitstellung von Verweisen auf die jeweils geänderten Entitäten. Diese Verweise wurden in Form von Paaren bestehend aus Identifikator und Zeichenkette modelliert, um Konflikte mit den übrigen Entitäten des Daten-

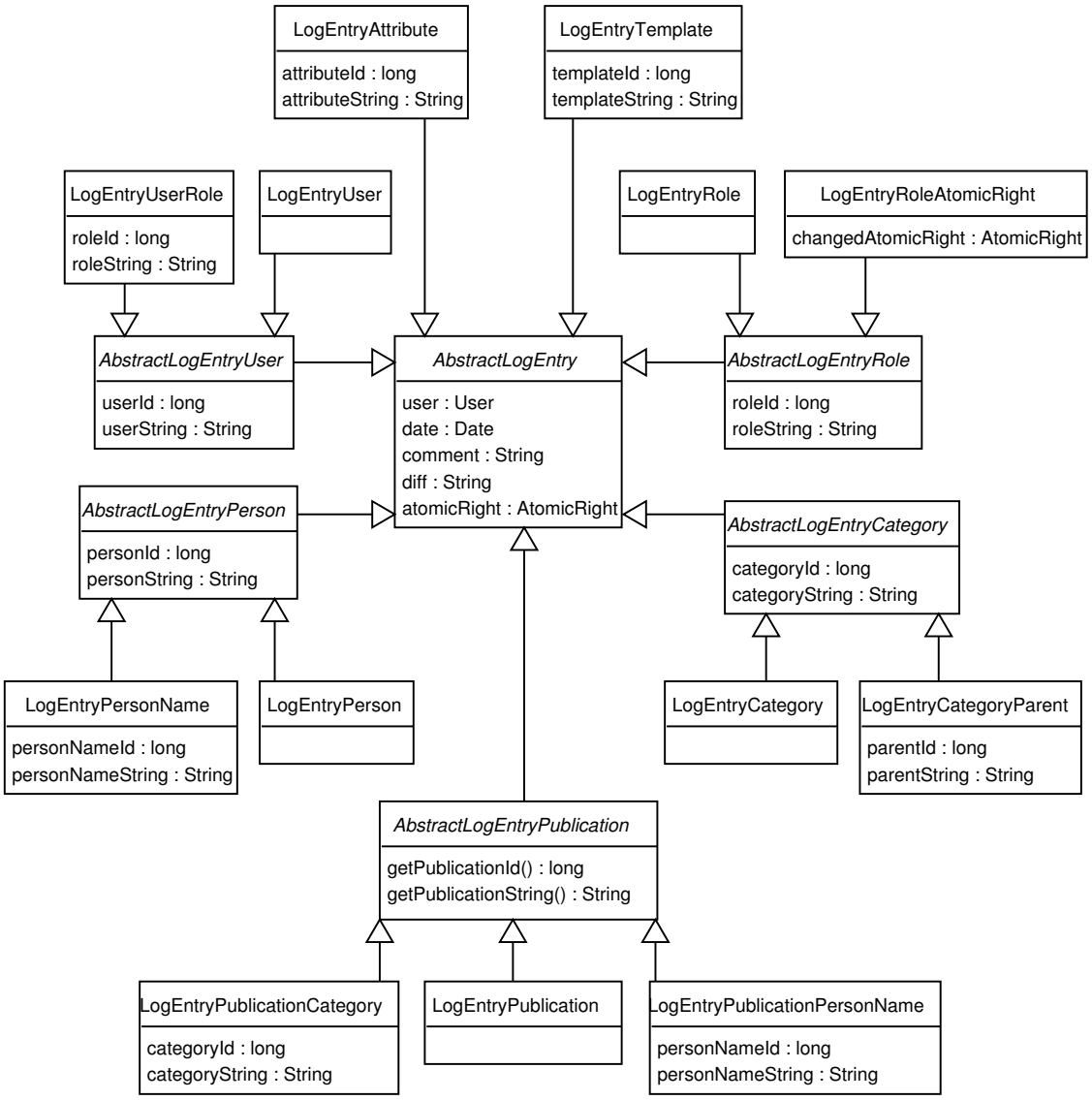


Abbildung 5.3.: UML-Klassendiagramm der Protokollierung

modells zu vermeiden. Die Modellierung der Verweise mittels gewöhnlicher Assoziationen und deren übliche Implementierung führt etwa beim Löschen von Entitäten auf die Problematik, dass entweder alle Protokolleinträge für die gelöschte Entität ihre Assoziation aufgeben müssen oder das physische Löschen von Entitäten mittels Statusfelder auf ein logisches Löschen abgeschwächt werden muss. Der in den abgebildeten Klassen gespeicherte Identifikator erlaubt dagegen eine lose Assoziation zwischen Protokolleintrag und bearbeiteter Entität. Nach Löschen einer Entität können dann über die menschenlesbare Zeichenkette zumindest weiterhin identifizierende Merkmale der Entität ausgegeben werden.

5.4. Verwaltung

Um administrative Aufgaben allgemeiner Natur darstellen zu können, wurde die in Abbildung 5.4 gezeigte Klasse `Task` konzipiert. Konkrete Verwendung findet diese Klasse etwa beim Melden von Konflikten während des Importvorgangs oder bei der Meldung von bibliographischen Unstimmigkeiten über die Weboberfläche der Anwendung.

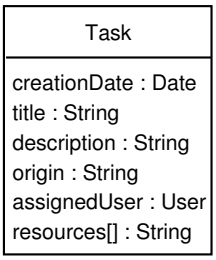


Abbildung 5.4.: UML-Klassendiagramm der Verwaltungsaufgaben

Neben einem kurzen Titel und Hinweisen auf Erstellung und Ursprung der Aufgabe kann im Wesentlichen ein längerer Freitext zur Problembeschreibung hinterlegt werden. Als Ergänzung können einer Aufgabe Referenzen auf die betroffenen Entitäten mittels URI-basierter Bezeichner zugeordnet werden, um den Zugriff und damit die Bearbeitung der Aufgabe zu beschleunigen.

6. Persistenz

Dieses Kapitel erläutert die Konzepte für das Arbeiten mit den persistenten Daten von oCIOBS. Dabei stehen einerseits Vorteile und Fallstricke der verwendeten Technologien und andererseits die sich daraus ergebenden Konsequenzen für das Design der Software im Vordergrund.

6.1. Hintergrund: EJB3 und Hibernate

Die Gruppe war sich einig, dass das Projekt so wenig wie möglich abhängig von einer bestimmten Technologie sein soll. Um dieses Ziel zu erreichen, ist es wichtig, sich an bestehenden Standards zu orientieren. In einem Application Server regelt *EJB3* unter anderem die Speicherung von *EntityBeans*. Als Nachfolger des *EJB2*-Standards vereinfacht es Konfiguration und Deployment der EntityBeans. Diese Beans repräsentieren persistente Daten, die beispielsweise in einer Datenbank gespeichert werden und damit unabhängig vom Laufzeitsystem sind.

Eine Implementierung dieses Standards bietet Hibernate (siehe Kapitel 3.2 auf Seite 19). Hibernate kann im Sinne des EJB3-Standards in einem Application Container benutzt werden, bietet dieselbe Funktionalität aber auch außerhalb eines Application Servers an.

Für das Datenmodell war es leider nötig, von den im Standard definierten Beschreibungen der Beans abzuweichen. Hibernate bietet etwas mehr Freiheit in der Abbildung der Datenklassen in die Datenbank. Um standardkonform zu bleiben, wären also Umbauarbeiten am schon vorhandenen Design nötig gewesen.

Ein weiterer Grund, direkt Hibernate zu benutzen, ist die Criteria-API (siehe Abschnitt 3.2.4 auf Seite 20). Für diese Technologie gibt es im EJB3-Standard keine Entsprechung. Trotzdem ist gerade diese Abfrage-„Sprache“ sehr nützlich und vor allem bei der Suche in den gespeicherten Daten hilfreich und nahezu notwendig.

Die Vorteile eines Einsatzes von Hibernate überwiegen also den Nachteil, dass es sich um eine proprietäre und damit nicht einfach austauschbare Lösung handelt. Um die einzelnen Module des Projektes trotzdem austauschbar zu halten, muss Hibernate also so gut wie möglich hinter einer weiteren Schicht versteckt werden. Diese kümmert sich dann auch um die Initialisierung beim Start von oCIOBS. Es gibt zwar beim JBoss-Application-Server die Möglichkeit, direkt mit Hibernate als Persistenzmanager zu arbeiten und mittels der gleichen Mechanismen, mit denen man den EJB3-Standard konfiguriert und nutzt, mit Hibernate-Objekten zu arbeiten. Dies

ist aber eine proprietäre Erweiterung, die nicht in Produkten anderer Hersteller funktioniert.

6.2. Datenabfrage

Alle Funktionen von Hibernate, die von anderen Modulen bezüglich des Ladens und Speicherns von Daten benötigt werden, müssen über die *Persistenzschicht* geführt werden. Deshalb sind einerseits Methoden nötig, die „einfache“ Funktionen kapseln. Das sind beispielsweise grundlegende Aktionen wie der Beginn oder das Ende einer Transaktion, oder direkte Suchanfragen. Andererseits dient diese Schicht dann der Ausführung von komplexen, häufiger benötigten Anfragen. Ein Beispiel dafür ist der sogenannte *Ko-Autor Index* (siehe Abschnitt 2.3.2 auf Seite 8). Die Information, mit wem eine Person schon gemeinsam Texte veröffentlicht hat, ist nicht direkt in der Datenbank gespeichert. Um diese Liste zu erhalten, ist eine komplexe Abfrage nötig, die aber vollständig von der Persistenzschicht versteckt wird.

Das Ändern von Daten an persistenten Objekten ist dank Hibernate transparent möglich. Änderungen an den persistenten Java-Objekten werden automatisch in die Datenbank übernommen. Zusätzlich wird die Transaktionsverwaltung mit explizitem Start und Ende von Hibernate auch von der Persistenzschicht übernommen, um atomare Schreibvorgänge in der Datenbank zu ermöglichen.

6.3. Erweiterte Suche

Um die Austauschbarkeit einzelner Komponenten auch bei der erweiterten Suche (siehe Kapitel 9 auf Seite 76) zu gewährleisten, wurde diese in zwei Teile geteilt. Der eine Teil analysiert die Suchanfragen, der andere arbeitet mit Hibernate zusammen, um die entsprechenden Daten anzufordern. Der Zugang zu den persistenten Daten muss über die Persistenzschicht erfolgen, um Konfigurations- und Caching-Probleme bei zwei nebeneinander laufenden Datenbankzugriffen zu vermeiden.

Dieser zweite Teil erzeugt spezielle SQL-Anfragen, die nicht direkt von der Persistenzschicht abgebildet werden können. Für die Suche musste also extra ein Weg geschaffen werden, an ein echtes SQLQuery-Objekt zu kommen. Dafür benutzen die Klassen das Wissen, das hinter der generischen Persistenzschicht Hibernate steckt, und bekommen so direkten Zugriff auf die Datenbank über SQL-Anfragen.

6.4. Architektur

Die endgültigen Beziehungen zwischen den einzelnen Modulen sind dann wie in Abbildung 6.1 auf der nächsten Seite dargestellt. Die Basis des Projekts ist die Datenschicht mit den Entity Beans. Diese wird in Kapitel 5 auf Seite 38 erläutert. Die

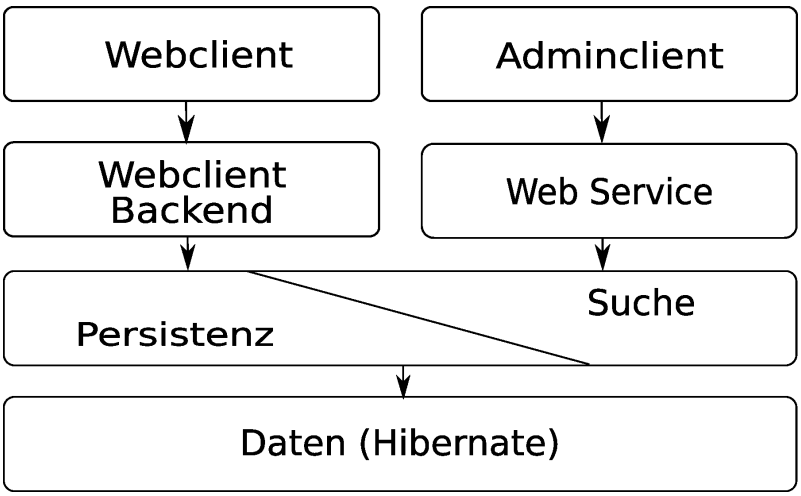


Abbildung 6.1.: Architektur des Projekts oCOBS

entstandenen Java-Klassen werden durch EJB3-Annotations größtenteils standardkonform beschrieben und von Hibernate verwaltet.

Die Persistenzschicht arbeitet direkt mit Hibernate zusammen und steuert das Anlegen und Abfragen von persistenten Objekten. Nach oben bietet sie Methoden, die genau diese Aktivitäten sowie direkte Suchanfragen kapseln. Das Suchmodul, welches Anfragen in der Suchsprache in SQL zur Abfrage von Objekten übersetzt, arbeitet über direkt mit einer Verbindung zur Datenbank. Es gehört also auch auf diese Ebene.

Der Webclient (siehe Kapitel 8 auf Seite 60) arbeitet immer im selben Application Container wie die Persistenz- und Datenschicht, da überflüssiger Datenaustausch zugunsten von Zugriffsgeschwindigkeit vermieden werden soll. Er greift also direkt auf eine Instanz der Persistenzschicht zu, die vom benutzten Application Server verwaltet wird. So entsteht kaum zusätzliche Verzögerung beim Abfragen der Daten aus der Datenbank.

Wie der Webclient arbeitet der Adminbackend-Web Service mit der Persistenzschicht. Er greift direkt auf die angebotenen Methoden zu und exportiert Dienste für Verwaltungstätigkeiten wie das Einstellen und Ändern von Veröffentlichungen. Außerdem bietet er eine öffentliche Schnittstelle für die Suche nach Objekten in oCOBS. So können alle angebotenen Dienste auch plattform- und technologieübergreifend genutzt werden.

Der Adminclient kommuniziert nur mit dem Web Service. Da hier die Geschwindigkeit und massive Parallelität der Zugriffe eine untergeordnete Rolle spielen, fiel die Entscheidung zugunsten der Nutzung einer allgemeinen Schnittstelle.

7. Import

Dieses Kapitel beschäftigt sich mit dem Import der Daten, die für die bisherige Version der DBLP genutzt werden. Diese Daten setzen sich im Wesentlichen aus den bibliographischen Datensätzen und dem bibliographischen Hypertext zusammen, die beide wiederum jeweils aus einer DTD und einer entsprechend validen XML-Datei bestehen (siehe Abschnitt 5.1 auf Seite 38).

Das Ziel des Imports ist die konsistente Überführung dieser XML-Dateien in das beschriebene Datenmodell, so dass die Daten von dem Web- und dem Adminclient genutzt werden können. Der Import stellt keinen einmaligen Prozess dar, sondern es sollen auch Aktualisierungen der lokalen Daten möglich sein. Bei diesen Aktualisierungen muss ein besonderes Augenmerk auf der Vermeidung von potenziellen Inkonsistenzen liegen.

Wie bereits angesprochen, besteht das zu importierende Datenmaterial im Wesentlichen aus zwei Teilen, wobei der bibliographische Hypertext die bibliographischen Datensätze referenziert und somit eine einseitige Abhängigkeit besteht. Daher wird nach der Vorstellung der technologischen Grundlagen zuerst der Import der bibliographischen Datensätze und darauf aufbauend der Import des Hypertextes dargestellt.

7.1. Technologien

Die Modellierung des Import-Prozesses erfolgt mit dem JavaABC (siehe Kapitel 3.7 auf Seite 27), während der Zugriff auf die XML-Daten über *StAX*[1] (siehe [12]) realisiert wird. Im Folgenden wird die Verwendung dieser Technologien im Rahmen des Imports erläutert.

7.1.1. JavaABC

Entsprechend dem Paradigma der *Lightweight Process Coordination* wird die klassische Drei-Schichten-Architektur um eine Koordinationsschicht erweitert, die zwischen der Geschäftslogik und der Präsentationsschicht eingefügt wird. Im Rahmen dieser Koordinationsschicht wird der Import-Prozess mit dem JavaABC (siehe Kapitel 3.7 auf Seite 27) modelliert.

[1] **St**reaming **A**PI for **X**ML

Dazu wird die Logik des Imports in Form von SIBs gekapselt, welche dann als grundlegende Bausteine zur Modellierung von Prozessen dienen. Die entstehenden SIB-Graphen können anschließend als GraphSIBs wiederum in andere Prozesse eingebunden werden, so dass eine konsistente und hierarchische Modellierung möglich ist. Dadurch kann der Import-Prozess auf oberster Ebene vergleichsweise grob modelliert werden, während die einzelnen Aktivitäten des Prozesses über eingebundene GraphSIBs wesentlich detaillierter abgebildet werden können.

Aus softwaretechnischer Perspektive erfolgt die Implementierung der SIBs, indem alle im Rahmen der Modellierung der Import-Prozesse verwendeten SIBs von der abstrakten Klasse `AbstractImportSIB` erben. Diese Basisklasse implementiert die Interfaces `Executable`, `CodeGeneratorBlock` sowie `LocalCheck` und definiert somit das Standardverhalten eines erbenden SIBs. Ein solches SIB implementiert im einfachsten Fall nur die Methode `trace` des *Tracer*-Plugins, mit dem die erstellen SIB-Graphen ausführbar gemacht werden. In dieser Methode des jeweiligen SIBs wird die zugehörige Geschäftslogik nicht direkt implementiert, sondern per *Reflection* (siehe [25]) an eine weitere Methode delegiert.

Aufgrund dieser Trennung zwischen der aufrufenden SIB-Klasse und der tatsächlichen Realisierung der Geschäftslogik ist es möglich, dass einerseits eine Modellierung mit den SIBs erfolgen kann, ohne dass die Implementierung der Geschäftslogik verfügbar ist. Andererseits können die implementierenden Methoden ohne Verwendung der Koordinationsschicht beziehungsweise der SIB-Klassen direkt aufgerufen und so beispielsweise getestet werden.

Die Implementierung dieser Geschäftslogik sowie die Speicherung des aktuellen Zustands des jeweiligen Import-Prozesses ist in Form einer zentralen Oberklasse `ImportContext` realisiert. Abhängig davon, ob es sich um den Import der bibliographischen Datensätze oder des bibliographischen Hypertextes handelt, wird eine Instanz der entsprechenden Spezialisierung dieser Oberklasse bei der Initialisierung des Prozesses in der Ausführungsumgebung des JavaABC abgelegt, so dass die Methoden dieser spezialisierten Kontext-Klasse im Folgenden per Reflection verwendet werden können.

Wie bereits beschrieben implementiert die abstrakte Basisklasse der SIBs auch das Interface `LocalCheck`, welches standardmäßig eine Reihe von Überprüfungen der lokalen Eigenschaften eines SIBs definiert, sowie das Interface `CodeGeneratorBlock`. Daher kann mittels des *CodeGenerator*-Plugins aus dem erstellten SIB-Graphen Java-Code erzeugt werden, der auch außerhalb der JavaABC Anwendung nur mit den Klassen des JavaABC Frameworks ausführbar ist.

Damit der so erstellte Code entsprechend der LPC von der Präsentationsschicht aufgerufen werden kann, wird er in Form einer Session EJB zur Verfügung gestellt.

7.1.2. StAX

Die *Streaming API for XML* ist eine Programmierschnittstelle zum Zugriff auf XML-Dokumente, die seit der JEE5 offizieller Teil der Java Enterprise Edition ist.

Obwohl StAX eine relativ neue Technologie ist, hat sich die Projektgruppe für deren Nutzung entschieden, da die beiden weitverbreitetsten APIs aus verschiedenen Gründen nicht den Anforderungen entsprachen:

- Die *DOM*[2]-API erstellt mit einem Aufruf direkt einen kompletten DOM-Baum vom zu parsenden Dokument im Speicher. Dies stellte sich auf Grund der puren Größe des zu verarbeitenden XML-Dokuments als problematisch heraus, auch wenn DOM einen sehr flexiblen Zugriff auf das XML-Dokument ermöglicht.

- *SAX*[3] hingegen verbraucht bei der Verarbeitung eines XML-Dokuments vergleichsweise wenig Arbeitsspeicher, jedoch läuft der Verarbeitungsvorgang dabei nach dem *Push*-Prinzip, was bedeutet, dass der Vorgang einmal angestossen wird und anschließend der SAX-Parser – abhängig von während des Parsens auftretenden Ereignissen – bestimmte *Callback*-Funktionen aufruft, die entsprechend für die Weiterverarbeitung des Ereignisses verantwortlich sind.

  Das Problem des Push-Prinzips ergibt sich in Verbindung mit der Nutzung des JavaABC, da während des Parsens die Steuerung über den Kontrollfluss vom Parser übernommen wird und somit der komplette Prozess des Parsens nicht mittels des JavaABC hätte modelliert werden können.

StAX arbeitet genau wie SAX ereignisbasiert und hat dementsprechend einen vergleichsweise niedrigen Speicherbedarf. Jedoch erfolgt der Vorgang des Parsens im Gegensatz zu SAX nach dem *Pull*-Prinzip. Das heißt, jedes Ereignis muss während der Verarbeitung explizit vom StAX-Parser angefordert werden, so dass die Steuerung des Kontrollflusses weiterhin der aufrufenden Methode – also im Grunde dem JavaABC – obliegt.

Der Zugriff auf die XML-Daten wird auf Basis der StAX-Technologie unter Verwendung des *Iterator-Entwurfsmusters* umgesetzt. Dabei werden die Informationen eines bibliographischen Datensatzes beziehungsweise eines Eintrags des Hypertextes durch folgende beiden Klassen gekapselt:

- `PublicationRecord` beinhaltet die Informationen genau eines bibliographischen Datensatzes in untypisierter Form, das heißt, es existieren nur explizite Zugriffsfunktionen für den eindeutigen DBLP-Bezeichner sowie für den Typ des Datensatzes, während alle anderen eingelesenen Felder in einer Map gespeichert sind, die als Schlüssel den Namen des entsprechenden XML-Elements und als Wert den Inhalt des XML-Elements enthält.

[2] **D**ocument **O**bject **M**odel
[3] **S**imple **A**PI for **X**ML

- `HypertextRecord` stellt die Kapselung eines Hypertext-Eintrages dar. Da der Hypertext in wesentlich unstrukturierterer Form als die bibliographischen Datensätze vorliegt, wird in dieser Klasse neben dem eindeutigen DBLP-Bezeichner und dem Typ des Eintrags keine Map der eingelesenen XML-Elemente verwaltet, sondern ein aus den geparsten StAX-Ereignissen erzeugter DOM-Baum des jeweiligen Eintrags. Somit ist es möglich, bei der weiteren Verarbeitung dieses Objekts mit Hilfe von *XPath* (siehe [30]) gezielt Informationen aus dem DOM-Baum zu extrahieren.

Instanzen dieser beiden Klassen werden beim Import von den Klassen `BibTexReader` beziehungsweise `HypertextReader` erzeugt und anschließend den verarbeitenden Methoden zur Verfügung gestellt. Dazu implementieren `BibTexReader` und `HypertextReader` das generische Interface `java.util.Iterator`, parametrisiert mit den Klassen `PublicationRecord` beziehungsweise `HypertextRecord`. So können nun entsprechend dem *Iterator-Entwurfsmuster* mit den Methoden `hasNext` und `next` bibliographische Datensätze und Hypertext-Einträge eingelesen werden, wobei jeweils erst beim Aufruf dieser Methoden versucht wird, einen neuen Datensatz beziehungsweise Eintrag einzulesen.

Auf Grund der Verwendung des Iterator-Musters ist somit ein vergleichsweise abstrakter Zugriff auf die eingelesenen XML-Daten bei der Implementierung der entsprechenden SIBs möglich.

7.2. Import der bibliographischen Datensätze

Das Ziel dieses Imports ist die Verarbeitung der bibliographischen Datensätze der DBLP, um daraus Entitäten des Datenmodells von oCOBS (siehe Abschnitt 5.1 auf Seite 38) zu erstellen. Ein einzelner Datensatz wird dabei in eine Entität des Typs `Publication` beziehungsweise `CompoundPublication` überführt. Zusätzlich werden noch die Angaben zu Autoren und Editoren ausgewertet, um daraus Instanzen von `Person` und `PersonName` zu erzeugen, die mit den entsprechenden Publikationen in Relation gesetzt werden.

Dieser Import-Prozess erfolgt auf oberster Modellierungsebene (siehe Abbildung 7.1 auf der nächsten Seite) im Wesentlichen in folgenden Schritten:

1. Zuerst erfolgt eine Initialisierung des gesamten Prozesses. Dabei wird unter anderem eine Instanz der Klasse `DblpPublicationContext` erzeugt und in der Ausführungsumgebung gespeichert, die benötigten XML-Daten der DBLP werden heruntergeladen und der `BibTexReader` wird initialisiert.

2. Anschließend erfolgt das iterative Einlesen und Verarbeiten der `BibTexRecords`, das näher im folgenden Abschnitt 7.2.1 auf der nächsten Seite beschrieben wird.

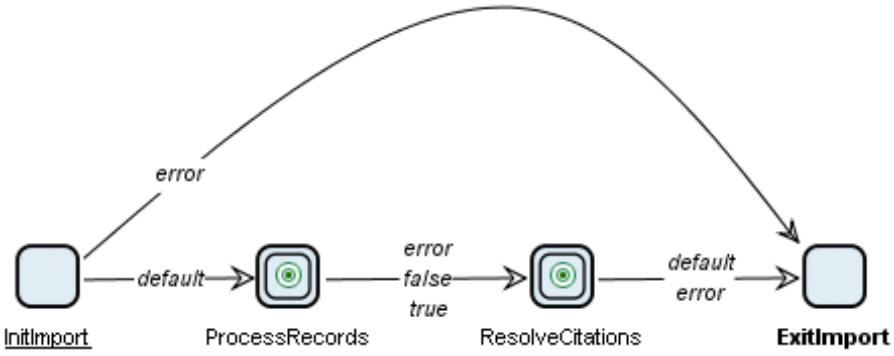


Abbildung 7.1.: Import von Publikationen

3. Nachdem alle Datensätze eingelesen wurden, werden die *citation links* (siehe Abschnitt 2.2.3 auf Seite 7) zwischen den einzelnen `Publications` gesetzt. Dies erfolgt anhand einer Map, in der während des vorherigen Schritts die Referenzen zwischen den einzelnen Datensätzen zwischengespeichert wurden.

7.2.1. Verarbeitung eines Datensatzes

Das Verarbeiten eines bibliographischen Datensatzes beginnt mit dem Einlesen eines `PublicationRecords`. Anhand des DBLP-Schlüssels dieses Datensatzes wird überprüft, ob sich eine `Publication` mit demselben Schlüssel bereits in der Datenbank befindet.

Falls dies zutrifft, werden die Zeitstempel der gefundenen `Publication` und des `PublicationRecords` miteinander verglichen, die angeben, wann der entsprechende Eintrag zuletzt geändert wurde.

Dabei können zwei Fälle auftreten:

- Sind die Zeitstempel identisch oder gibt der Zeitstempel der `Publication` ein neueres Datum an, müssen keine weiteren Aktionen durchgeführt werden und es kann mit der Verarbeitung des nächsten Datensatzes begonnen werden.

- Falls aber der Zeitstempel des `PublicationRecords` ein neueres Datum angibt, ist der bibliographische Datensatz in den XML-Daten der DBLP geändert worden und somit nicht mehr konsistent mit den Daten, die zu einem vorherigen Zeitpunkt in oCOBS importiert wurden.

  Ein einfaches Übernehmen der neuen Daten aus der DBLP ist nicht möglich, da so eventuell vorhandene, lokale Änderungen an der `Publication` überschrieben würden. Daher wird ein `Task`-Objekt erzeugt, das den Administrator darüber informiert, dass eine entsprechende Modifikation in den Daten der DBLP aufgetreten ist. Der Administrator hat dann die Möglichkeit, die

entsprechenden Änderungen manuell einzupflegen. (siehe Abschnitt 13.4 auf Seite 108)

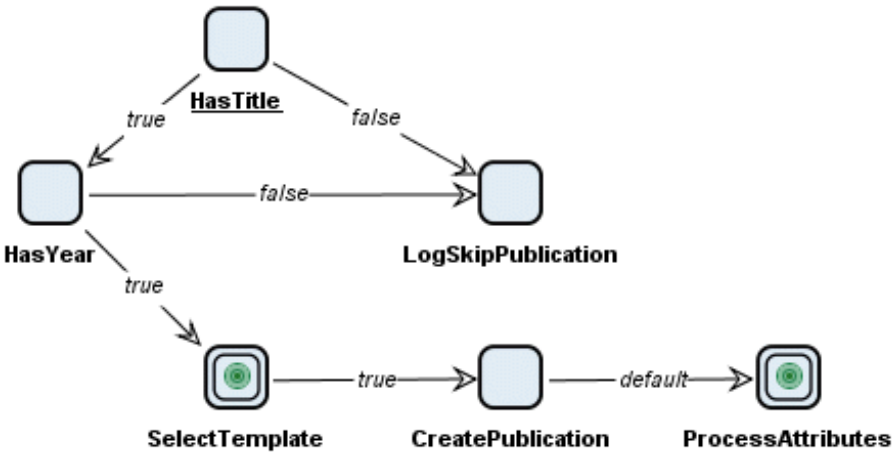


Abbildung 7.2.: Neue Publikation erzeugen

Für den Fall, dass keine `Publication` unter dem entsprechenden DBLP-Schlüssel in der Datenbank gefunden wurde, muss eine neue `Publication` angelegt werden.

Der zugehörige Ablauf ist in der Abbildung 7.2 dargestellt und wird im Folgenden näher beschrieben:

- Zuerst wird überprüft, ob der eingelesene `PublicationRecord` die Attribute `year` und `title` besitzt. Falls diese nicht vorhanden sind, wird die aktuelle Datensatz übersprungen und mit der Bearbeitung des nächsten Datensatzes begonnen.
- Anschließend wird anhand des Attributes `type` des `PublicationRecords` das entsprechende `PublicationTemplate` gewählt.
- Mithilfe des ausgewählten Templates kann dann ein neues `Publication`-Objekt erzeugt werden, das jedoch bis auf den DBLP-Schlüssel und Default-Werte für Jahr und Titel, die im nächsten Schritt wieder überschrieben werden, keine weiteren Attribute enthält.
- Im letzten Schritt erfolgt die Auswertung der Attribut-Map des `PublicationRecord` und die Erstellung der Listen von Werten für die entsprechenden `Attributes` (siehe Abschnitt 5.1 auf Seite 38).

  Eine Sonderrolle spielt dabei die Verarbeitung der Autoren und Editoren, da in diesen Fällen das richtige `PersonName`-Objekt zu einem String mit dem Namen ermittelt werden muss. Das Vorgehen dabei wird in dem Abschnitt 7.2.2 auf der nächsten Seite erläutert.

7.2.2. Autoren und Editoren

Bei der Verarbeitung von Autoren und Editoren während des Imports sind zwei Aspekte von Bedeutung: einerseits muss die im Rahmen des Abschnitts 5.1 auf Seite 38 angesprochene Problematik der Identifikation von Personen mittels ihres Namens berücksichtigt werden, andererseits muss eine Aufteilung des Namens in Titel, Vor- und Nachname durchgeführt werden.

Zur Aufspaltung der Namen kommt eine Heuristik zum Einsatz. Im Rahmen dieser Heuristik werden die Namens-Strings zuerst in ihre einzelnen Token zerlegt. Anschließend wird jedes Token auf Basis von Listen über verbreitete Vor- und Nachnamen des östlichen bzw. westlichen Kulturkreises, regulären Ausdrücken zum Erkennen von akademischen Titeln, typischen Namenssuffixen und -präfixen und der Position des Tokens innerhalb des Namens klassifiziert. Die dabei eingesetzten Namenslisten wurden anhand von allgemeinen Statistiken über häufige Personennamen (siehe [33] und dortige Querverweise) sowie der Analyse häufiger Namensteile speziell im Bestand der DBLP gewonnen.

Darauf aufbauend wird bestimmt, ob es sich um einen Namen aus dem östlichen oder aus dem westlichen Kulturkreis handelt. Dies ist von Bedeutung, da bei östlichen Namen im Gegensatz zu westlichen Namen der Familienname vor dem Vornamen notiert wird.

Aus den so gewonnenen Informationen kann abschließend die Aufspaltung des Namens in einen optionalen Titel, Vor- und Nachname durchgeführt werden.

Das generelle Vorgehen beim Feststellen eines `PersonName`-Objekts zu einem gegebenen Namen gestaltet sich so, dass zuerst überprüft wird, ob zu dem gegebenen Namen bereits eine passene Kombination aus `PersonName` und `Person` beziehungsweise ein passendes `Person`-Objekt, zu dem noch ein `PersonName` hinzugefügt werden muss, vorhanden ist.

Falls keine passenden Objekte in der Datenbank gefunden wurden, kann das entweder den Grund haben, dass zu der gesuchten Person noch gar keine Informationen in der Datenbank vorhanden sind, oder, dass die Person bereits in der Datenbank vorhanden ist, aber bisher nur unter einem anderen Namen.

Um dies festzustellen, wird anhand des gegebenen Namens der URL der Autorenseite ermittelt und die entsprechende HTML-Seite heruntergeladen. Wenn beim Herunterladen der Autorenseite ein Redirect auf eine andere Autorenseite zurückgegeben wird, steht fest, dass die Person bereits unter einem anderem Namen in der Datenbank vorhanden ist, und der entsprechenden Instanz von `Person` wird ein neues `PersonName`-Objekt hinzugefügt. Andernfalls werden die Informationen aus der heruntergeladenen Autorenseite verwendet, um ein neues `Person`-Objekt zu erzeugen, und diesem neu erzeugten Objekt wird dann der `PersonName` hinzugefügt.

Das Problem, dass mehrere Personen den gleichen Namen tragen, wird in der DBLP gelöst, indem mehrere Personen mit dem gleichen Namen intern durch das

Anhängen einer vierstelligen Nummer[4] voneinander unterschieden werden.

Diese Zusatzinformation in Form von Suffixen wird auch beim Import genutzt, um Personen mit dem gleichen Namen voneinander zu unterscheiden.

7.3. Import des bibliographischen Hypertextes

Dieser Import-Prozess beschäftigt sich mit den Einträgen des bibliographischen Hypertextes, die in Entitäten des Datenmodells von oCOBS überführt werden, um die Navigationsstruktur und Kategorisierung der DBLP nachzubilden.

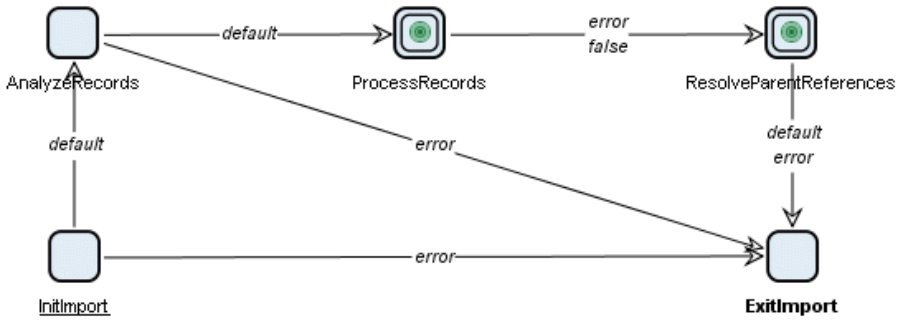


Abbildung 7.3.: Import von Kategorien

Der oberste SIB-Graph für den Import des Hypertextes (siehe Abbildung 7.3) ist analog zu dem Import der bibliographischen Datensätze aufgebaut. Jedoch werden vor der eigentlichen Verarbeitung der Hypertext-Einträge die XML-Daten bereits einmal durchlaufen, um Informationen auszuwerten und im `CategoryContext` zu speichern, die in den folgenden Verarbeitungschritten benötigt werden. Darunter fallen unter anderem die Extraktion von Titeln für Kategorien vom Typ *ConferenceSeries* und das Ermitteln der Proceedings, die in mehrere Unterkategorien aufgeteilt sind.

Analog zum Import der bibliographischen Datensätze erfolgt auch nach der Verarbeitung der einzelnen Hypertext-Einträge das Setzen der Relationen zwischen den Kategorien, die während der vorherigen Phase in dem `CategoryContext` zwischengespeichert wurden.

7.3.1. Verarbeitung eines Eintrags des bibliographischen Hypertextes

Die Abbildung 7.4 auf der nächsten Seite beschreibt den Ablauf bei der Verarbeitung eines Hypertext-Eintrags. Der erste Schritt dabei stellt die Verzweigung anhand

[4] siehe zum Beispiel `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/h/Hofmann_0002:Martin.html`

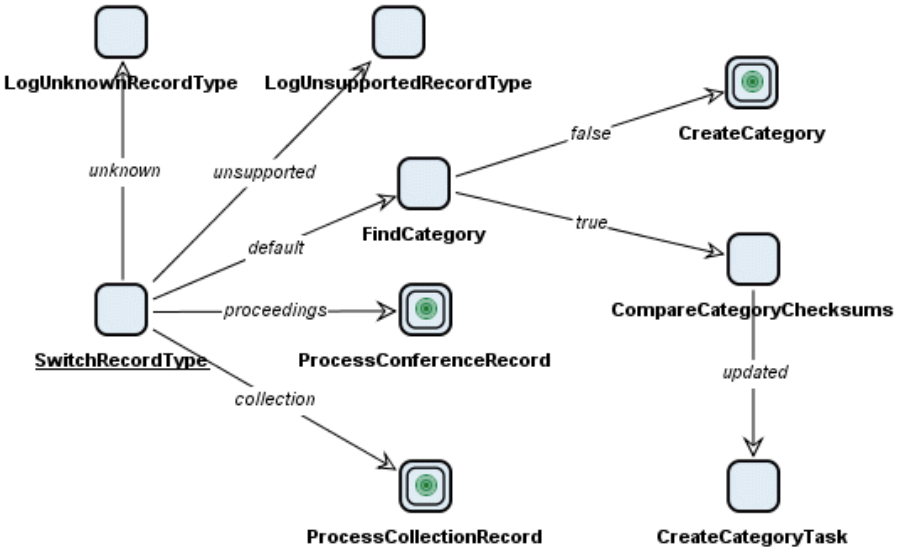


Abbildung 7.4.: Verarbeitung eines Hypertext-Eintrags

des Typs des `HypertextRecords` dar. Dieser Typ wird beim Parsen des Eintrags abhängig vom Wert des jeweiligen DBLP-Schlüssels bestimmt und beschreibt unter anderem, welcher der vier höchsten Kategorien, *Conferences*, *Journals*, *Collections* und *Series*, der aktuelle Eintrag zuzuordnen ist.

Wie sich an den Kanten der Verzweigung erkennen lässt, existieren auch Typen von `HypertextRecords`, deren Verarbeitung bisher noch nicht unterstützt wird oder dessen DBLP-Schlüssel nicht erkannt werden und somit keine Typisierung zulassen. (siehe Abschnitt 13.3 auf Seite 108)

In einem Großteil der Fälle wird jedoch die *default*-Kante gewählt und es folgt die Überprüfung, ob eine `Category` mit dem Schlüssel des aktuellen `HypertextRecords` bereits existiert. Sofern dies zutrifft, werden die Prüfsummen der gefundenen `Category` und des `HypertextRecords` miteinander verglichen, um festzustellen, ob die XML-Daten der DBLP modifiziert wurden, und gegebenenfalls den Administrator mithilfe eines neuen `Task`-Objekts davon in Kenntnis zu setzen.

Die angesprochene Prüfsumme wird benötigt, da im Gegensatz zu den bibliographischen Datensätzen für die Hypertext-Einträge kein Attribut auf Seiten der DBLP-Daten existiert, was das Datum der letzten Änderung dokumentiert. Daher wird beim Parsen eines Eintrags mittels des Adler32-Algorithmus[5] eine Prüfsumme über die Elemente des DOM-Baums berechnet.

Falls bei der obigen Überprüfung keine `Category` gefunden wurde, wird nun eine neue `Category` angelegt. Dazu wird zuerst das entsprechende `CategoryTemplate` ausgewählt und damit eine einfache `Category` mit dem Titel und dem DBLP-

[5] siehe `http://java.sun.com/j2se/1.5.0/docs/api/java/util/zip/Adler32.html`

Schlüssel des Hypertext-Eintrags erzeugt. Dabei wird außerdem die Prüfsumme des Hypertext-Eintrags in die erzeugte Kategorie übernommen.

Anschließend werden abhängig vom Typ der Kategorie weitere Methoden aufgerufen, um die Attribute und Relationen zu anderen Kategorien beziehungsweise Publikationen auszuwerten. Außerdem werden in diesen Methoden Unterkategorien zur weiteren Gliederung erzeugt; so werden beispielsweise im Rahmen der Verarbeitung einer Kategorie vom Typ *JournalVolume* in der Regel weitere Unterkategorien der Typen *JournalIssue* und *JournalSubject* erstellt, in denen die Verweise auf die einzelnen Artikel gespeichert sind.

7.3.2. Spezialfälle bei der Verarbeitung von Hypertext-Einträgen

Das Datenmodell (siehe Abschnitt 5.1 auf Seite 38) lässt es zu, dass `Publications` nicht nur als Blatt-Knoten in der Navigationsstruktur der Kategorien eingebunden werden, sondern in Form von `CompoundPublications` auch Unterkategorien besitzen können.

Diese Eigenschaft von `CompoundPublications` wird beim Import von Hypertext-Einträgen der Typen *Conference* und *Collection* benötigt.

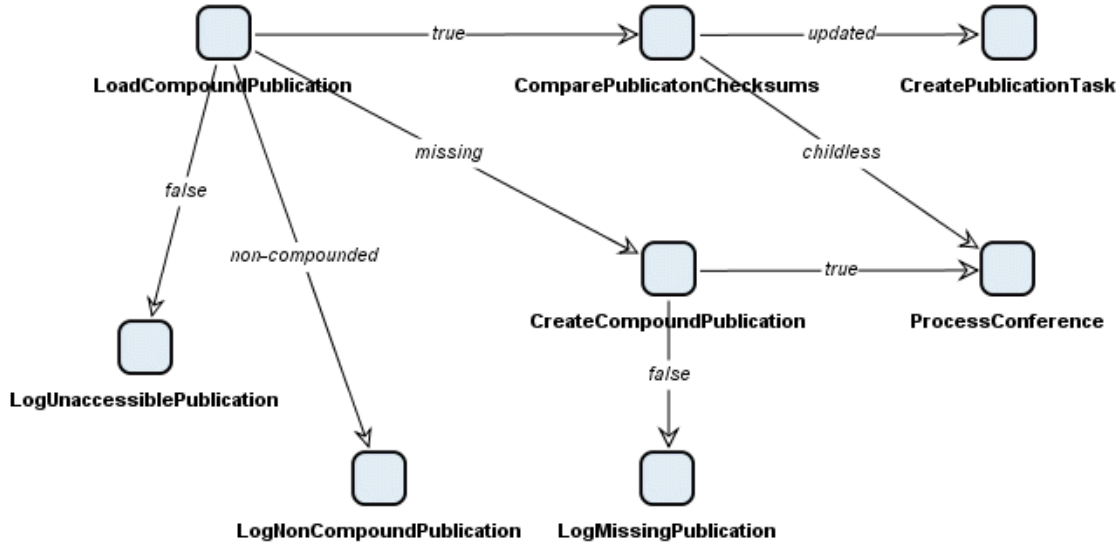


Abbildung 7.5.: Verarbeitung eines Hypertext-Eintrags des Typs *Conference*

Das Vorgehen bei derartigen Hypertext-Einträgen wird im Folgenden anhand des Typs *Conference* erläutert (siehe 7.5), wobei die Verarbeitung von Einträgen des Typs *Collection* analog erfolgt.

Im ersten Schritt dieser Verarbeitung wird versucht, den DBLP-Schlüssel des Proceedings zu ermitteln, der zu der aktuell betrachteten Konferenz gehört. Dazu wird die Struktureigenschaft von Hypertext-Einträgen des Typs *Conference* genutzt, dass

sich an einer bestimmten Stelle des DOM-Baums ein Verweis auf das zugehörige Proceedings befindet, wenn dieses Proceedings Teil der bibliographischen Datensätze ist.

Abhängig davon, ob ein entsprechender Verweis vorhanden ist, lassen sich folgende Fälle unterscheiden:

- Ist dieser Verweis vorhanden, wird im Normalfall die entsprechende `CompoundPublications` des Typs *Conference* korrekt geladen und die Prüfsummen des geladenen Proceedings und des `HypertextRecords` werden miteinander verglichen. Anders als bei dem im vorherigen Abschnitt beschriebenen Prüfsummenvergleich ist es hierbei auch von Bedeutung, ob die gefundene `CompoundPublication` bereits Unterkategorien besitzt. Aufgrund dessen lassen sich folgende vier Fälle für den Ausgang des Prüfsummenvergleichs identifizieren:

  - Falls das Proceedings weder Unterkategorien noch eine Prüfsumme besitzt, befindet es sich zum ersten Mal an dieser Stelle des Import-Prozesses und kann weiterverarbeitet werden.
  - Falls bereits Unterkategorien existieren, aber keine Prüfsumme vorhanden ist, wurden dem Proceedings manuell Unterkategorien hinzugefügt und es findet keine weitere Verarbeitung des Hypertext-Eintrags statt.
  - Die weitere Verarbeitung des aktuellen `HypertextRecord` kann außerdem beendet werden, wenn eine Prüfsumme vorhanden ist und diese mit der des Hypertext-Eintrags übereinstimmt.
  - In dem Fall, dass die Prüfsummen nicht übereinstimmen, wurden die DBLP-Daten verändert und der Administrator wir mithilfe eines `Task`-Objekts über die Änderung informiert.

- Falls der Verweis auf das Proceedings nicht vorhanden ist, wird aus dem DBLP-Schlüssel des `HypertextRecords` ein standardisierter Schlüssel erzeugt, mithilfe dessen dann nach einer entsprechenden `CompoundPublication` gesucht wird. Wird ein entsprechendes Proceedings gefunden, läuft der Verarbeitungsprozess wie im ersten Fall mit dem Prüfsummenvergleich weiter.

  Andernfalls wird eine neue `CompoundPublication` mit dem standardisierten Schlüssel und mit den Informationen des Hypertext-Eintrags erstellt, da für den weiteren Prozess ein entsprechendes Proceedings benötigt wird. Anschließend kann der Eintrag weiterverarbeitet werden.

Im weiteren Prozess werden analog zur Verarbeitungs von *JournalVolumes* die Unterkategorien und dazugehörigen InProceedings bestimmt und entsprechend als Kinder der `CompoundPublication` hinzugefügt.

8. Webclient

Der *Webclient* stellt die Benutzerschnittstelle zu den verwalteten bibliographischen Daten dar. Über einen Browser kann der Benutzer Publikationen und Autoren suchen. Dabei stehen ihm unterschiedliche Möglichkeiten der Suche zur Verfügung. Im Hintergrund werden die angeforderten Daten aus der Datenbank gelesen und weitere für die Anzeige benötigte Daten generiert. Anschließend können die Daten im Browser angezeigt werden.

Des Weiteren ist der Zugriff auf die Publikationen durch Auswahl der jeweiligen Publikationsorgane möglich.

Ein grundlegender Aspekt bei der Entwicklung des Webclients ist die Performanz. Im produktiven oCOBS-Einsatz ist mit einer hohen Anzahl paralleler Zugriffe zu rechnen. Aus diesem Grund werden im Frontend ausschließlich die grundlegenden Technologien *JSP* und *Servlets* der Java Enterprise Edition eingesetzt statt Frameworks wie *JSF*. Dem höheren Entwicklungsaufwand steht eine entscheidend bessere Leistungsfähigkeit gegenüber. Die ausgelieferten Seiten sind in der Regel bedeutend kleiner als JSF-Äquivalente.

Im Folgenden wird zunächst der Zugriff auf den Webclient aus der Sicht des Benutzers beschrieben. Hier werden die unterschiedlichen Möglichkeiten der Navigation durch die bereitgestellten Publikationen aufgezeigt. Die anschließenden Abschnitte beschreiben die technische Realisierung. Hierbei werden zunächst allgemeine Konzepte vorgestellt, welche bei allen Benutzerzugriffen zum Einsatz kommen. Darauf folgen detaillierte Beschreibungen der Realisierung der Einzelfunktionalitäten.

8.1. Benutzerzugriff auf den Webclient

Der Webclient bildet grundsätzlich die existierende Struktur der *DBLP* ab. Die Seiten sind den bereits bestehenden in der Regel nachempfunden (siehe hierzu „Benutzerzugriff auf die Bibliographie“ auf Seite 7).

Einige Funktionalitäten wurden jedoch abgeändert und andere neu hinzugefügt. Die zur Verfügung stehenden Seiten und ihre Funktionen werden im Folgenden einzeln beschrieben. Der gesamte Webclient ist dabei sprachenunabhängig gestaltet. Momentan kann der Benutzer zwischen deutscher und englischer Anzeige wählen. Weitere Sprachen können jederzeit hinzugefügt werden.

8.1.1. Startseite

Die *Startseite* ist der Hauptzugangspunkt der Anwendung. Von hier aus besteht die Zugriffsmöglichkeit auf alle logischen Bereiche der Anwendung. Der Benutzer kann über die Navigation zu den einzelnen Bereichen (Startseite, Bibliographie, Suche, Merkzettel, Hilfe) gelangen. Außerdem werden auf der Startseite bereits einige Funktionen zur Verfügung gestellt. Es kann direkt eine Suche gestartet werden. Zudem wird der aktuelle Server-Status mit der Angabe der aktuellen Besucheranzahl und der Anzahl der im System erfassten Autoren und Publikationen angezeigt. Wie auf jeder anderen oDOBS-Seite befindet sich auch hier ein Hilfe-Block, welcher an dieser Stelle allgemeine Hilfsinformationen zum oDOBS-System enthält.

Abbildung 8.1 zeigt die Startseite von oDOBS.

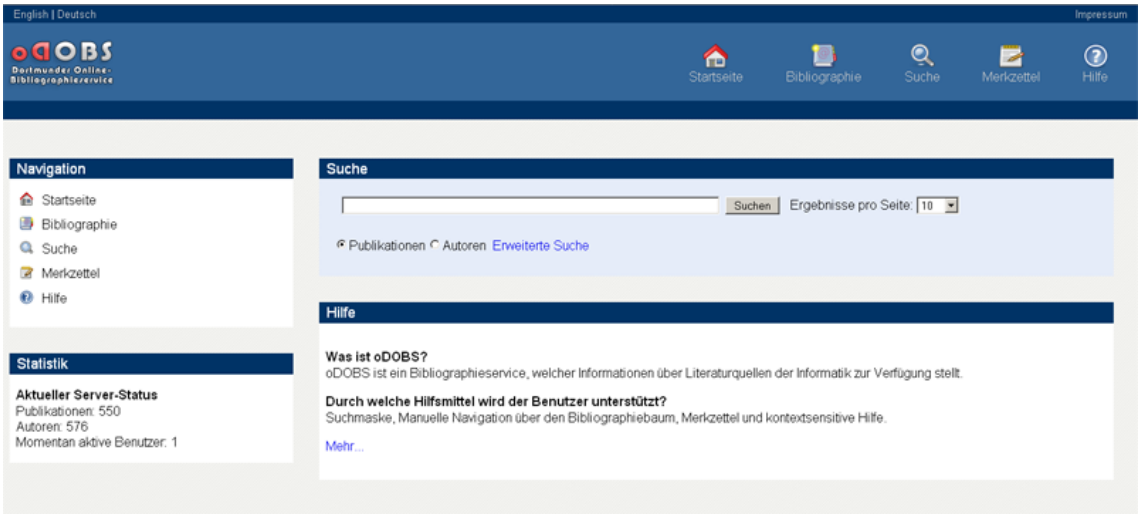


Abbildung 8.1.: Startseite

8.1.2. Autorenseite

Auf einer *Autorenseite* werden dem Benutzer alle zur Verfügung stehenden Informationen zu einem Autor angezeigt.

Abbildung 8.2 auf der nächsten Seite zeigt einen Ausschnitt einer Autorenseite in oDOBS.

Sind in der Datenbank – neben den Publikationen und den Namen eines Autors – weitere Informationen gespeichert, so werden diese am Anfang der Seite angezeigt. In vielen Fällen enthält die Datenbank zum Beispiel einen Link zu der persönlichen Homepage des Autors.

Bei der Auflistung der Publikationen unterscheidet sich die Anzeige, je nachdem, ob die Person Autor, Editor oder sowohl Autor als auch Editor von Publikationen ist.

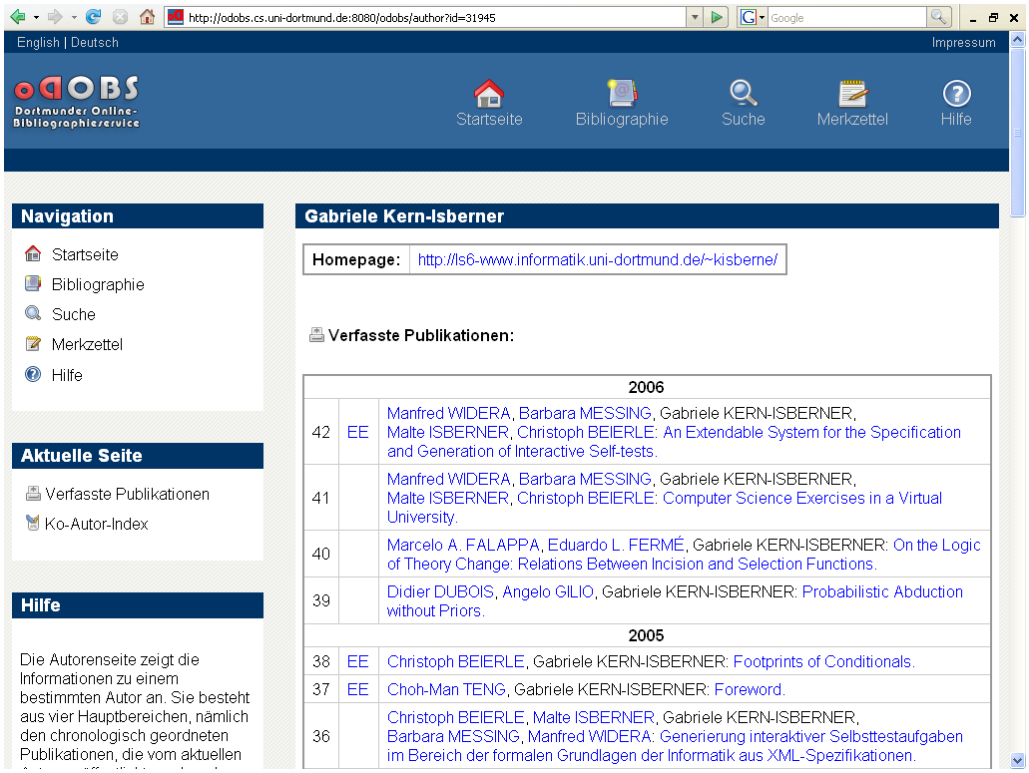


Abbildung 8.2.: Ausschnitt einer Autorenseite in oDOBS

Im ersten Fall wird eine Liste der publizierten Publikationen und der Ko-Autor-Index angezeigt. Ist die Person nur Editor, so ist eine Liste der editierten Publikationen und der Ko-Editor-Index zu sehen. Ist die Person jedoch sowohl Autor als auch Editor wird zunächst die Liste der publizierten Publikationen und der Ko-Autor-Index und anschließend der Liste der editierten Publikationen und der Ko-Editor-Index angezeigt.

Die Listen von Publikationen sind dabei nach Erscheinungsjahr absteigend sortiert und gruppiert. Jeder Listeneintrag enthält neben dem Titel der Publikation eine Liste aller Autoren und Editoren (in kursiver Schrift). Mit einem Klick auf den Publikationstitel gelangt der Benutzer zu der jeweiligen *Publikationsseite* (siehe hierzu Kapitel 8.1.3 auf der nächsten Seite). Ebenso kann über die Namen der Autoren beziehungsweise Editoren zu den einzelnen Autorenseiten gesprungen werden. Zu den einzelnen Publikationen wird – neben einer mit dem Erscheinungsjahr aufsteigenden Nummerierung – über dem Wort EE[1] der Link zu der elektronischen Version der Publikation angebeben, sofern dieser im Datenbestand erfasst ist.

Unter der Liste der verfassten beziehungsweise editierten Publikationen wird jeweils der aus Kapitel 2.3.2 auf Seite 8 bekannte *Ko-Autor-Index* beziehungsweise

[1]EE steht für Electronic Edition

Ko-Editor-Index angezeigt.

Die Aufteilung der Publikationen in verfasste und editierte Publikationen, sowie die Auftrennung in Ko-Autor- und Ko-Editor-Index stellen eine echte Erweiterung gegenüber der DBLP dar.

Unter der Überschrift „Aktuelle Seite“ wird dem Benutzer angezeigt, welche Elemente (Liste von verfassten Publikationen und Ko-Autor-Index oder Liste von editierten Publikationen und Ko-Editor-Index) die aktuelle Seite enthält. Außerdem kann mittels Sprungmarken innerhalb der Seite zu den einzelnen Seitenelementen gesprungen werden. Aufgrund von beispielsweise Heirat oder Namensänderungen kann es vorkommen, dass ein Autor unter verschiedenen Namen publiziert hat. Ist dies der Fall, wird dem Benutzer des Webclients ebenfalls unter der unter Überschrift „Aktuelle Seite“ eine Liste der zu dem Autor gehörenden Namen angezeigt.

Wie auf jeder Seite wird auf der Autorenseite auch ein Hilfe-Block angezeigt, der Erklärungen zur Autorenseite enthält.

8.1.3. Publikationsseite

Auf einer *Publikationsseite* werden alle zu einer Publikation gehörenden Informationen dargestellt. Dazu gehört neben dem Titel und dem Jahr der Veröffentlichung eine Auflistung der Autoren beziehungsweise Editoren. Die hier aufgelisteten Namen sind mit den jeweiligen Autorenseiten verlinkt.

Darauf folgt – sofern der Datenbank bekannt – eine Auflistung der referenzierten und referenzierenden Artikel. In dieser Auflistung sind neben dem Titel der Publikation die jeweiligen Autoren und das Jahr der Veröffentlichung zu sehen. Im weiteren Teil dieser Tabelle werden zusätzliche Informationen zu der Publikation angezeigt. Diese können je nach Datenbestand sehr unterschiedlich umfangreich sein.

Zu den im Block mit der Überschrift „Aktuelle Seite“ aufgeführten Funktionen auf der Publikationsseite gehören das Hinzufügen der Publikation in den Merkzettel (siehe Abschnitt 8.1.6 auf Seite 65), sowie der Export der Publikation in eines der drei unterstützten Formate *BibTeX*, *PDF* und *HTML* (siehe Abschnitt 8.1.7 auf Seite 65). Dort kann ebenfalls die Anzahl der sich bereits auf dem Merkzettel befindenden Publikationen entnommen werden. Außerdem wird hier angezeigt, in welchen Kategorien sich eine Publikation befindet.

Publikationsformen wie Sammel- oder Konferenzbände werden als zusammengesetzte Publikationen *CompoundPublication* verstanden. Auf einer Publikationsseite enthalten sie einen weiteren Block unter den allgemeinen Informationen zur Publikation. Dort werden die der Publikation zugehörigen Kategorien und Publikationen angezeigt.

8.1.4. Bibliographie/ Kategorieseite

Neben der Suche nach Artikeln über bestimmte Suchkriterien ist der alternative Zugriff auf Publikationen über den Bibliographiedienst möglich. Hierbei beginnt der Benutzer mit einem Klick auf den Button „Bibliograhie“. Zunächst werden alle Wurzelkategorien angezeigt. Da die Wurzelkategorien viele Kinderkategorien haben können, kann der Benutzer sich entweder alle Kinderkategorien einer Wurzelkategorie anzeigen lassen oder lediglich alle Kinderkategorien mit einem bestimmten Anfangsbuchstaben. Wählt der Benutzer eine bestimmte Kategorie aus, so werden ihm auf der Kategorieseite in einer Tabelle alle zur Kategorie gespeicherten Informationen angezeigt. Zudem erhält er eine Übersicht der Kinderkategorien und Publikationen der jeweiligen Kategorie.

Unter der Überschrift „Aktuelle Seite“ kann der Benutzer die jeweiligen Elterkategorien[2] oder die `CompoundPublication` sehen, zu welcher die Kategorie gehört.

8.1.5. Suchseite

Die Struktur der *Suchseite* beruht auf den Standards, welche aus der Praxis in Bezug auf Ästhetik und Benutzerfreundlichkeit von führenden Suchmaschinenbetreibern herausgearbeitet worden sind. Ganz oben platziert sich eine Suchmaske. Es werden zwei Modi unterstützt: erweiterter Modus und einfacher Modus. Im einfachen Modus entscheidet der Benutzer anhand der Radiobuttons, ob er nach Publikationen oder nach Autoren sucht. Die fortgeschrittenen Nutzer des Webclients können in diesem Modus komplexe Suchbefehle eingeben (siehe Abschnitt 9.3.1 auf Seite 82). Ein Nutzer ohne Vorkenntnisse, dem die einfache Suche nicht ausreicht, kann die erweiterte Suche verwenden (siehe Abschnitt 9.3.2 auf Seite 82). Sie stellt mehrere Felder zur Verfügung, um die Suchergebnisse besser einschränken zu können. Unter der Suchmaske befindet sich die Ergebnisliste. Die Ergebnisse werden als Liste dargestellt. Die wichtigsten Felder eines Datensatzes werden angezeigt. Die Autoren und Publikationen sind wiederum mit den Autoren- beziehungsweise Publikationsseiten verlinkt. Es besteht die Möglichkeit, das Suchergebnis nach bestimmten Kriterien absteigend oder aufsteigend zu sortieren. Die Auflistung der Sortierungskriterien wird vor der Suchergebnisausgabe angezeigt. Weiterhin befindet sich am Seitenanfang eine Auswahlbox um einzustellen, wie viele Ergebnisse pro Seite angezeigt werden sollen. Zwischen den einzelnen Ergebnisseiten kann über Navigationslinks gewechselt werden. Nähere Informationen zu den Suchfunktionen sind in Abschnitt 9 auf Seite 76 zu finden.

[2] Eine Kategorie kann prinzipiell mehrere Elterkategorien haben.

8.1.6. Merkzettel

Der Benutzer hat die Möglichkeit, Publikationen in einem „Warenkorb“, *Merkzettel* genannt, zu speichern. Diese Funktion dient der einfachen Zusammenstellung von Literaturlisten.

Jede auf einer Ergebnisseite oder in der Bibliographie angezeigte Publikation verfügt über einen Link, um diese zum persönlichen Merkzettel hinzuzufügen. Es ist jederzeit möglich, eine Übersicht über alle aktuell auf dem Merkzettel gespeicherten Publikationen anzuzeigen. Zur besseren Übersicht werden bei dieser nur Titel und Erscheinungsjahr angezeigt. Soll eine Publikation vom Merkzettel entfernt werden, kann dies mittels eines Löschlinks durchgeführt werden. Es ist auch möglich den Merkzettel zurückzusetzen, also alle enthaltenen Publikationen zu entfernen.

8.1.7. Export

Publikationen können sowohl von Publikationsseiten, als auch vom Merkzettel aus exportiert werden. Beim Merkzettel besteht dabei die Möglichkeit die gesamte zusammengestellte Liste zu exportieren. Als Formate stehen *BibTeX*, *PDF* und *HTML* zur Verfügung.

8.1.8. Hilfe

Für die Unterstützung der Benutzer und zur Beantwortung möglicher Fragen zur Systemnutzung stehen `Hilfe`-Seiten zur Verfügung. Es gibt eine zentrale `Hilfe`-Seite, welche beim Auswählen des Menü-Eintrags `Hilfe` angezeigt wird. Von hier aus besteht die Möglichkeit auf einzelne Hilfebereiche überzugehen.

8.1.9. Statistiken

Die *Startseite* ist im Moment die einzige Stelle, wo Statistiken angezeigt werden. Dabei handelt es sich um einen Block, in welchem die Statistiken zum Serverstatus zu sehen sind. Es gibt keine separate *Statistik*-Seite, wie es bei anderen Funktionalitäten der Fall ist.

8.2. Grundlagen der Realisierung

Nachdem in Kapitel 8.1 auf Seite 60 der Benutzerzugriff auf den Webclient beschrieben wurde, erläutern die folgenden Kapitel die technische Realisierung der einzelnen Komponenten.

8.2.1. Verarbeitungsprozess

Damit im Browser Daten aus der Datenbank angezeigt werden können, muss die Anfrage des Benutzers im Browser über mehrere Schichten an die Datenbank gestellt werden. Anschließend werden die gefundenen Daten in aufbereiteter Form an den Browser übermittelt. Der Verarbeitungsprozess besteht aus vier Schichten. Anfragen beginnen grundsätzlich in der Clientebene – dem Browser. Von hier aus werden Servletaufrufe gestartet, welche an den – sich in der nächsten Schicht befindenden – *Tomcat* Webserver gerichtet sind. Auf dieser Schicht werden Anfrageparameter ausgewertet und entsprechende Methoden der *Session Beans* aufgerufen. Die Session Beans befinden sich auf der *JBoss-Application-Server*-Ebene und beinhalten die Geschäftslogik. Für viele Anfragen werden Daten aus der Datenbank benötigt, welche sich in der untersten Schicht befindet. Zur Realisierung der Schnittstelle zwischen den Session Beans und der Datenbank ist eine *Persistenzschicht*[3] entwickelt worden, um den Datenbankzugriff zu kapseln. Diese befindet sich zwischen der Session Bean und *Hibernate*, welches direkte Anfragen an die Datenbank stellt. Die Session Bean verarbeitet die zurückgelieferten Daten und stellt diese dem Servlet zur Verfügung. Das Servlet befüllt eine oder mehrere *JavaBeans* mit den für die Anzeige im Browser notwendigen Objekten. Die im Browser angezeigte *JSP*-Seite kann nun auf die in der JavaBean hinterlegten Daten zurückgreifen und diese anzeigen.

Abbildung 8.3 auf Seite 74 stellt beispielhaft für die Anzeige einer Autorenseite im Browser dar, über welche Schichten die Benutzeransicht mit der Datenbank verbunden ist.

8.2.2. Layout

Um einen modularen Seitenaufbau zu ermöglichen, werden bei jedem Servletaufruf die Komponenten einer Layout-Bean gesetzt. Die Layout-Bean speichert die folgenden Parameter:

- `Title`
  Titel der aktuellen Webseite.
- `Header`
  Url zum Block, welcher im Webseitenkopf angezeigt wird.
- `Left`
  Liste mit den Urls, welche auf die auf der linken Seite anzuzeigenden Blöcke verweisen.
- `Main`
  Liste mit den Urls, welche auf die in der Mitte anzuzeigenden Blöcke verweisen.

[3] Siehe hierzu auch Kapitel 6 auf Seite 46

Der Servletaufruf endet mit der Speicherung der Layout-Bean im *Request-Scope* und der Weiterleitung zu `layout.jsp`. Diese JSP-Seite liest die gesetzten *URIs* und den Seitentitel aus der im Request-Scope gespeicherten Layout-Bean aus und inkludiert mittels der Direktive `jsp:include` die angegebenen Seiten in *div-Tags* und stellt diese modular aufgebaute Seite dar. Um einem Designer eine große Flexibilität in Fragen der Seitengestaltung ohne Vorkenntnisse der *JSP-Technologie* gewährleisten zu können, wurden alle Formatierungselemente aus den *JSP-Templates* herausgenommen. Die Darstellung ist komplett über *Stylesheet*-Dateien steuerbar. Abbildung 8.4 auf Seite 75 stellt am Beispiel einer Autorenseite dar, wie Servlets, JSP-Seiten und Beans für die Seitendarstellung zusammenarbeiten.

Im ersten Schritt wählt der Benutzer einen Link, welcher hier beispielsweise auf `AuthorServlet` verweist und HTTP-GET-Attribute, wie etwa `AuthorID`, enthält. `AuthorServlet` stellt fest, dass es direkt aufgerufen wurde und dass es nicht nur für die Inhaltslieferung, sondern auch für die Zusammensetzung der Layout-Bean verantwortlich ist. Im nächsten Schritt wendet sich `AuthorServlet` an die darunter liegende SessionBean, an `RequestAuthorBean`. Es werden einige Methoden der SessionBean aufgerufen, welche die Geschäftslogik einkapseln. Die SessionBean führt ihrerseits keine direkte Kommunikation mit der Datenbank durch, sondern verwendet die Persistenzschichtmechanismen, die SQL-Anfragen stellen.

Letztendlich erfragt `AuthorServlet` ein Objekt der Klasse `Person`, welches den aktuell gesuchten Autor repräsentiert. Zudem werden einige weitere Anfragen gestellt (beispielsweise für die Anzeige des *Ko-Autor-Index*) und alle Informationen über den Autor in `AuthorBean` gesammelt. Dieses JavaBean wird in den Request-Scope gelegt, von wo aus alle Daten der Darstellung zur Verfügung stellen. Das Servlet leitet anschließend an `layout.jsp` weiter. Die JSP-Seite lädt die Informationen aus den JavaBeans und stellt die Autorenseite im Browser dar.

8.2.3. Filter

In der Applikation werden die folgenden Servlet-Filter eingesetzt:

- ExceptionFilter
- LocaleFilter
- ReferrerFilter
- SessionFilter

ExceptionFilter

Um ein einheitliches Fehlermanagement zu gewährleisten, werden im Webclient Ausnahmen vom Typ `ODOBSException` produziert und verarbeitet. Der

`ExceptionFilter` hat die Aufgabe diese Ausnahmen herauszufiltern und das `ErrorServlet` anzustossen. Normalerweise würde für diese Aufgabe das Element `error-page` mit dem `exception-type`-Mapping auf die Klasse `ODOBSException` im Deployment-Descriptor `web.xml` genügen. Es kann allerdings vorkommen, dass eine `ODOBSException` gekapselt in einer `ServletException` auftritt. In diesem Falle muss der `ExceptionFilter` die gekapselte `ODOBSException` mittels der Methode `getRootCause()` extrahieren.

LocaleFilter

Der `LocaleFilter` realisiert die Sprachenunabhängigkeit des Webclients. Jede internationalisierte Textanzeige greift auf eine `LocaleBean` zu. Dieser Filter überprüft, ob in der *HTTP-Session* eine entsprechende Bean vorhanden ist und legt wenn nötig eine an. Diese wird dann mit den gewünschten Spracheinstellungen aus einem Cookie beziehungsweise – falls kein Cookie existiert – den übertragenen Spracheinstellungen aus dem Browser des Benutzers initialisiert.

ReferrerFilter

Wird die aktuell ausgewählte Sprache gewechselt oder eine Publikation dem Merkzettel hinzugefügt, muss die Applikation nach der Abarbeitung des Aufrufs wieder zur ursprünglichen Adresse weiterleiten. Dazu speichert der `ReferrerFilter` die URI des aktuellen Aufrufs in der HTTP-Session, so dass die Applikation zu dieser URI weiterleiten kann.

SessionFilter

Auch Aufrufe statischen Inhalts öffnen HTTP-Sessions, obwohl diese nicht benötigt werden. Der `SessionFilter` prüft bei jeder Anfrage, ob statischer Inhalt angefragt wird. Ist dies der Fall, wird die HTTP-Session wieder geschlossen.

8.3. Realisierung Einzelfunktionalitäten

Nachfolgend wird die Realisierung der jeweiligen Seiten und der einzelnen Funktionen des Webclients beschrieben.

8.3.1. Startseite

Eine charakteristische Eigenschaft aller bis jetzt implementierten Servlets ist, dass sie Methoden zum Aufbereiten des Inhalts sowohl für einen Block als auch für die Hauptdarstellung beinhalten. Die `Startseite` ist die einzige Ausnahme von dieser Regel. Dies erklärt sich dadurch, dass das hinter der `Startseite` stehende Servlet

(`IndexServlet`) niemals indirekt aufgerufen wird. Aus diesem Grund gibt es keine `indexBlock` und `indexMain` JSP-Fragmente, da diese trivialerweise überflüssig sind. Die einzige Aufgabe des `IndexServlet`s ist die Befüllung der Layout-Bean mit den Verweisen auf andere Servlets (`SearchServlet`, `BibliographyServlet` usw.), die einzelne Blöcke auf der Startseite generieren.

8.3.2. Autoren- und Publikationsseite

Eine Vielzahl der realisierten Funktionalitäten der Autorenseite wurde bereits im Kapitel 8.2 beschrieben. Das Grundkonzept zur Verwirklichung der Publikationsseite unterscheidet sich zudem kaum von dem der Autorenseite. Es werden lediglich andere *JavaBeans* in den *Request-Scope* gelegt, um den jeweiligen JSP-Seiten als Datenhalter zu dienen. Außerdem werden andere Servlets benutzt, um Daten für die Darstellung aufzubereiten.

Die zugehörigen Servlets der Seiten lesen den Parameter `ID` aus der Anfrage-URL aus. Die gelesene *ID* gibt eindeutig an, über welchen Autor beziehungsweise über welche Publikation die erforderlichen Daten aus der Datenbank angefragt werden sollen.

In den beiden JSP-Seiten (`author.jsp` und `publication.jsp`) werden eine Reihe geschachtelter *for-each Schleifen* benutzt, welche aus den einzelnen *JavaBeans* mehrere Daten gleichen Typs lesen. So werden beispielsweise bei `author.jsp` zu jedem Namen eines Autors (erstes for-each) alle geschriebenen Publikationen zu diesem Namen (zweites for-each) und alle Namen der Koautoren (drittes for-each) zu der jeweiligen Publikation ausgelesen. *for-each Schleifen* sind an dieser Stelle geeignet, da sie, falls sie auf eine leere Liste angewendet werden, einfach übersprungen werden und keine Ausgaben erzeugen.

Um von einer Autoren- oder Publikationsseite zu einer anderen Seite diesen Typs zu verlinken, wird von der JSP-Seite die *ID* des Autors oder der Publikation aus der *JavaBean* gelesen und ein Link gesetzt. Das angegebene Servlet kann nun wiederum aus der Anfrage-URL die gesetzte *ID* auslesen und wie oben beschrieben fortfahren.

Für die Darstellung der Publikationsseite einer *CompoundPublication* muss zusätzlich die Liste der zu einer Publikation gehörenden Publikationen und Kategorien erstellt werden. Dazu wird im `PublicationServlet` die rekursive Methode `addToCatelicationList`[4] aufgerufen, welche alle „unterhalb“ der Publikation befindlichen Kategorien und Publikationen in eine Liste einfügt. In `publication_compound.jsp` wird die Liste geeignet ausgelesen und dargestellt.

Für die unter der Überschrift „Aktuelle Seite“ dargestellten Informationen wird jeweils eine eigene JSP-Seite verwendet. Die Seiten tragen die Namen `author_menu.jsp` und `publication_menu.jsp`.

[4] CatelicationList steht hier für eine Liste aus Kategorien (Cate...) und Publikationen (...lication)

8.3.3. Bibliographie/ Kategorieseite

Für den Bibliographiedienst werden unterschiedliche JSP-Seiten (beginnend mit `category...`) verwendet. `CategoryServlet` fragt wie bei der Autoren- und Publikationsseite die erforderlichen Daten für die Darstellung einer Kategorieseite oder der Wurzelkategorienseite an und stellt den JSP-Seiten die Informationen über den Request-Scope zur Verfügung. Bei der Wurzelkategorieseite wird zur Filterung der Kinderkategorien über den Anfangsbuchstaben eine Liste der Anfangsbuchstaben im Servlet erzeugt. Neben dem Parameter `ID` kann zur Filterung der Kinderkategorien der Wurzelkategorien über den Anfangsbuchstaben der Parameter `beginswithchar` gesetzt werden. Dieser wird vom Servlet ausgelesen, so dass nur noch Kinderkategorien mit dem entsprechenden Anfangsbuchstaben auf der Kategorieseite angezeigt werden.

8.3.4. Suchseite

Eine detaillierte Beschreibung der Funktionalität der Suche findet sich in Kapitel 9 auf Seite 76. Im Folgenden wird der Prozess der Weiterleitung der Suchanfrage an die *SessionBean* und die anschließende Darstellung der Ergebnismenge behandelt.

Wie jede andere Komponente erzeugt das für die Generierung der Suchseite verantwortliche Servlet (`SearchServlet`) einen Block-Inhalt und einen Inhalt für die Hauptdarstellung. Im ersten Fall wird auf `searchBlock.jsp` zugegriffen und eine Suchmaske generiert, welche aus einem einzigen Textfeld und zwei Radiobuttons für die Unterscheidung zwischen der Suche nach Autoren und nach Publikationen besteht. Der entsprechende Parameter `item` wird über HTTP GET übergeben. Wird die Suche aus dem Block gestartet, erfolgt eine Weiterleitung auf die Hauptseite der Suche, welche mit Hilfe des `searchMain`-Fragments generiert wird. Oben auf dieser Seite befindet sich die Suchmaske im einfachen Modus. Der Modus wird vom Servlet über den Parameter `mode` gesteuert. Das Suchfeld enthält den eingegebenen Suchbegriff, der aus dem Parameter `searchString` ausgelesen wird.

Unter der Suchmaske platziert sich die Suchergebnismenge. Sie wird von `SearchServlet` ermittelt, indem ein *JNDI Lookup* durchgeführt wird und die Suchanfrage an die Methode `search` der EJB `SearchBean` weitergeleitet wird. Diese Methode wird mit einigen Angaben parametrisiert, welche bestimmen, welcher Teil der gesamten Suchergebnismenge dem Benutzer angezeigt werden soll. Insbesondere gibt es einen Parameter, der *Treffer pro Seite* angibt. Der Default-Wert ist 10. Der Wert wird jedoch in einem Cookie gespeichert, damit der Benutzer seinen bevorzugten Wert nicht bei jedem Besuch neu einstellen muss. Wenn die Anzahl an Suchtreffern größer als die gewünschten Treffer pro Seite ist, taucht vor und falls der Parameterwert größer als 10 ist, auch nach der Ergebisliste eine Leiste auf, die dem Benutzer das Navigieren zwischen den einzelnen Trefferseiten ermöglicht. Dazu gibt

es die Parameter `startIndex` und `resultsPerPage`. Sowohl im Block als auch auf der Hauptseite der Suche besteht die Möglichkeit zur erweiterten Suche zu wechseln. Dabei verändert sich die Form der Suchmaske. Wenn die Suche von der erweiterten Suchmaske gestartet wurde, wird automatisch zur einfachen Suche gewechselt und der von der erweiterten Suche zusammengesetzte Experten-Suchbefehl wird im Suchfeld angezeigt. Der Hauptgrund der Suchmoduswechsel ist die Größe der erweiterten Suchmaske. Deswegen werden dabei die Suchergebnisse im einfachen Suchmodus angezeigt. Will der Benutzer wieder zur erweiterten Suche wechseln, um die Suchparameter zu ändern, muss er erneut den entsprechenden Link wählen. Die für die letzte Suche eingestellten Suchparameterwerte werden wieder angezeigt.

8.3.5. Merkzettel

Die Funktionen *Hinzufügen einer Publikation*, *Entfernen einer Publikation* und *Anzeigen des Merkzettels* sind durch das `MemosServlet` mittels des übergebenen Parameters `do` realisiert. Es wird eine Liste mit den IDs aller zum Merkzettel des Benutzers gehörenden Publikationen verwaltet. Diese kann durch die Methoden `addMemo(String key)`, `removeMemo(String key)` und `clearMemos()` manipuliert werden. Zusätzlich wird die Liste mittels eines Cookies lokal beim Benutzer gespeichert.

Publikationen hinzufügen Durch Aufruf von `MemosServlet` mit den Parametern `do=add&key=id` wird die Publikation mit der Datenbank-ID `id` zur Liste des Merkzettels hinzugefügt. Das Servlet ruft hierzu die Methode `addMemo(String key)` der *EJB* auf. Ist noch keine Merkzettelliste vorhanden, wird eine neue Liste erstellt und die ID hinzugefügt. Im Anschluss belegt das Servlet die Attribute der `LayoutBean` und leitet zu `layout.jsp` weiter.

Publikationen löschen Durch Aufruf von `MemosServlet` mit den Parametern `do=remove&key=id` wird die Publikation mit der Datenbank-ID `id` aus der Liste des Merkzettels entfernt. Verwendet wird dazu die Methode `removeMemo(String key)` der EJB. Diese entfernt die ID aus der Liste der gemerkten Publikationen. Wird `MemosServlet` mit dem Parameter `do=remAll` aufgerufen, wird die Liste durch Aufruf der EJB-Methode `clearMemos()` geleert. Es werden alle vorhandenen Publikationen nach vorheriger Sicherheitsabfrage vom Merkzettel entfernt.

Anzeigen des Merkzettels Für die Anzeige des Merkzettels wird eine Liste von `Publication`-Objekten[5], die alle Publikationen des Merkzettels enthält, in der EJB generiert. Diese Liste wird über das Servlet in einer *JavaBean* gespeichert. Aus

[5] siehe Kapitel 5 auf Seite 38

dieser Bean werden im Seitenfragment `MemosBlock.jsp` Titel und Erscheinungsjahr gelesen und angezeigt.

8.3.6. Export

Der *Export* wurde mit Hilfe des *JavaABC*s modelliert. Gestartet wird der Export durch das `ExportServlet` mit den Parametern `format` und `id`. Diese geben das gewünschte Exportformat und die ID der zu exportierenden Publikation an. Diese ID kann auch den Wert `all` enthalten, wodurch alle Publikationen des Merkzettels exportiert werden.

Der Export besteht aus acht *SIB*s. Der Vorgang startet mit `ExportInit`. Dieses SIB initialisiert Parameter für im Exportvorgang benötigte Pfade und überprüft die übergebenen Parameter. Darauf folgt die Erstellung einer Bibliographie-Datei mit den zu exportierenden Publikationsdaten. Dabei werden soweit vorhanden standard BibTeX Benennungen der Attribute verwendet. Im Datenbestand werden in Publikationstiteln zum Teil HTML-Tags verwendet um beispielsweise hoch- oder tiefgestellten Text zu ermöglichen. Diese Tags werden, falls das gewählte Exportformat nicht HTML ist, durch ihre LaTeX-Äquivalente ersetzt. Realisiert werden diese Schritte mittels des `ConvertPublications`-SIB. Das Ergebnis ist eine standardkonforme Bibliographie-Datei, die im Fall des Exports im BibTeX-Format durch `CreateBibtex` gelesen und im Browser angezeigt wird.

Der Export in die Formate PDF und HTML ist durch externe Programme realisiert. Im Falle des PDF-Formates wird *pdflatex* und im Falle des HTML-Formates *latex2html* verwendet. Dadurch kann eine standardisierte Ausgabe der Attribute gewährleistet werden. Als *Style* wird `gerplain` verwendet. Dazu wird zunächst eine tex-Datei in der vom Nutzer eingestellten Sprache durch das SIB `CreateTex` erstellt. Im Anschluss daran werden die externen Prozesse durch `CreateHtml` beziehungsweise `CreatePdf` gestartet und das Ergebnis an den Browser gesendet. Sofern die Standardpfade der externen Programme nicht korrekt sind, müssen diese in der oCOBS-Konfiguration eingestellt werden. Für den Export im HTML-Format ist zu beachten, dass die vom `latex2html`-Prozess im temporären Verzeichnis erstellte Stylesheetdatei vom Client aus unter Umständen nicht erreichbar ist. Aus diesem Grund wird die Datei vor der Ausgabe eingelesen und die Style-Definitionen in den Kopfbereich der HTML-Ausgabe eingebunden.

Um einen Exportvorgang korrekt abzuschließen, wird das SIB `ExportExit` ausgeführt. Dieses hat die Aufgabe alle temporär angefallenen Dateien zu entfernen.

Tritt im Verlauf des Exports ein Fehler auf, wird dieser durch das SIB `ExportError` behandelt. Dadurch werden angemessene `ODOBSExceptions` generiert und temporäre Daten ebenfalls entfernt.

8.3.7. Hilfe

Die `Hilfe`-Komponente ist einschichtig implementiert. Das heißt, dass sie lediglich über das `HelpServlet` und zwei JSP-Seiten implementiert ist. Die Inhalte der Hilfe-Komponente werden nicht aus der Datenbank gelesen, sondern sind statisch in den ResourceBundles(in den Dateien `odobs_i18n_de.properties` und `odobs_i18n_en.properties`) der Anwendung hinterlegt. Die Verwirklichung der kontextabhängigen Hilfe ist durch den Parameter `action` möglich. Dieser Parameter muss von allen Servlets gesetzt werden, welche für die Belegung der Attribute der `LayoutBean` sorgen. Wenn zum Beispiel das `SearchServlet` eine `LayoutBean`-Instanz erzeugt, wird dem Parameter `action` der Wert `search` zugewiesen. Dadurch erkennt das `HelpServlet` den Kontext und entsprechende `Hilfe`-Inhalte werden erzeugt.

8.3.8. Statistiken

Die Implementierung der `Statistiken` ist an das allgemeine Konzept angelehnt, allerdings mit einer leichten Abweichung. Es gibt ein `StatisticsServlet`, eine `StatisticsBean` und zwei JSP-Seiten, nämlich `statisticsGeneralBlock.jsp` und `statisticsLastPublicationsBlock.jsp`. Die Abweichung vom allgemeinen Konzept besteht darin, dass `StatisticsSevlet` nicht direkt aufgerufen wird. Die `LayoutBean`-Instanz verweist auf eine der beiden JSP-Seiten, die mit einem gesetzten `action`-Parameter die Anfrage an das Servlet weiterleitet. Diese Maßnahme ist unumgänglich, wenn zwei oder mehrere Blöcke einer Komponente auf einer Seite angezeigt werden sollen. Dies ist bei den oCIOBS-Statistiken der Fall. Bis jetzt ist auf der Startseite nur die Anzeige der Server-Statistiken zu sehen. Bereits implementiert ist auch die Anzeige der zuletzt veröffentlichten Publikationen. Auf dieselbe Weise könnten auf der Startseite in Zukunft auch die vom `StatisticsServlet` erzeugten Server-Nachrichten angezeigt werden.

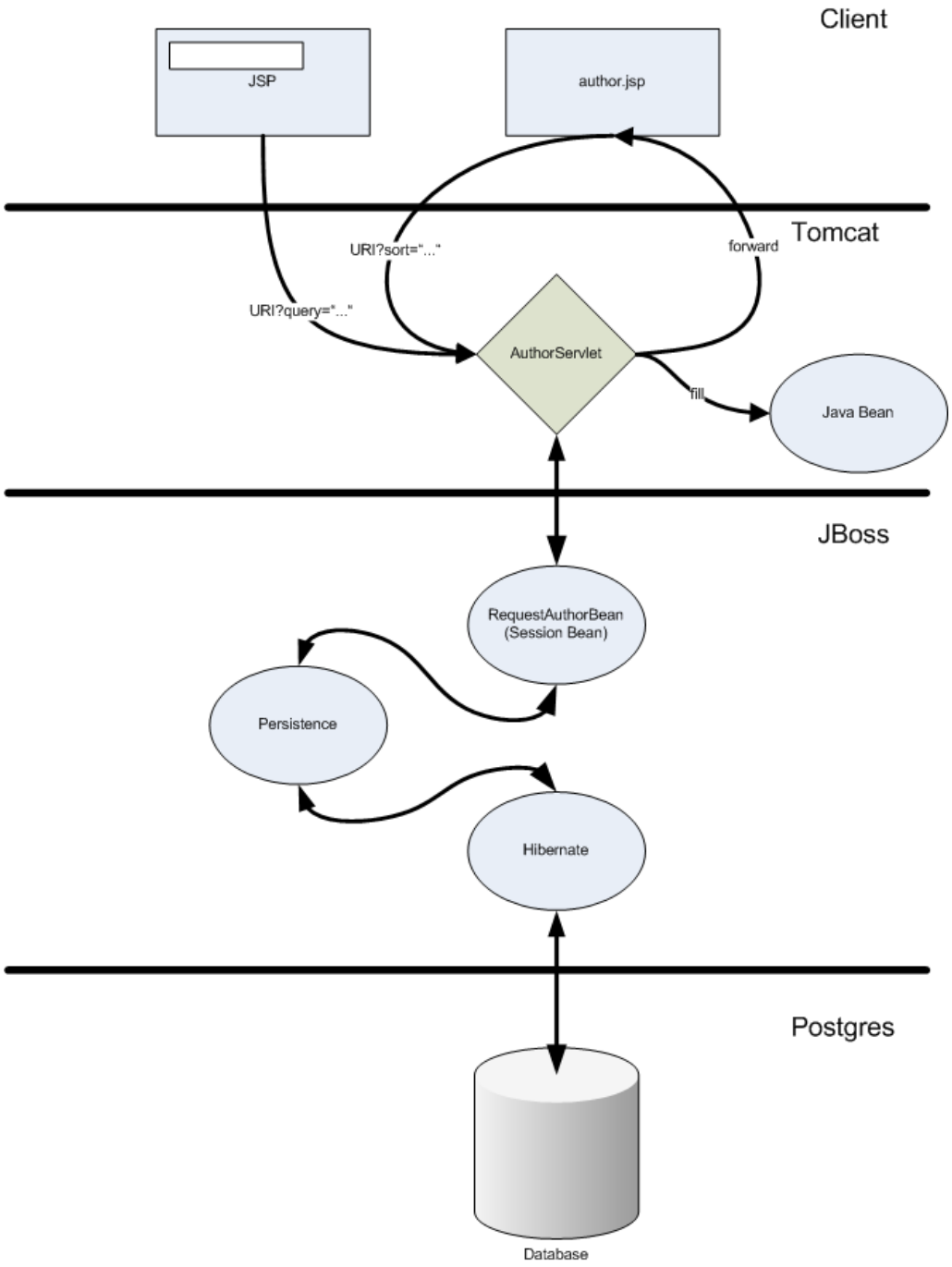


Abbildung 8.3.: Verarbeitungsprozess beim Aufruf einer Autorenseite

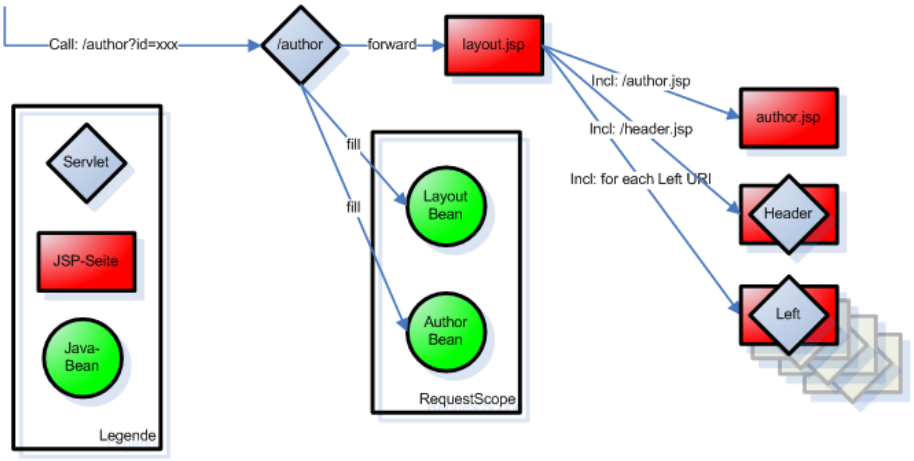


Abbildung 8.4.: Modularer Seitenaufbau beim Aufruf von `AuthorServlet`

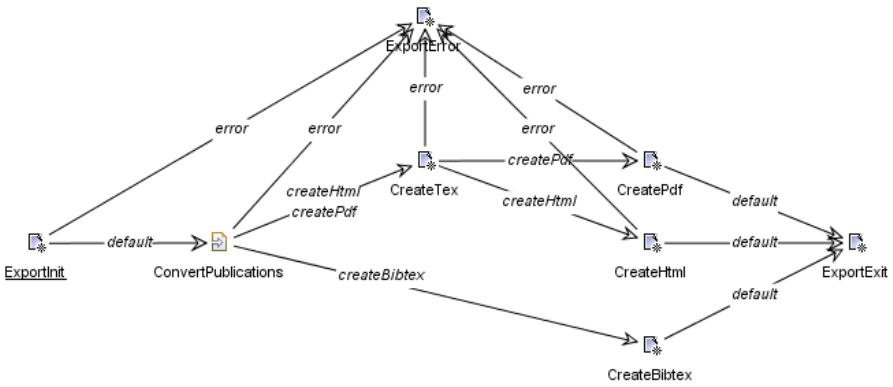


Abbildung 8.5.: Export JavaABC Modell

9. Suchfunktion

Eines der Hauptziele der Projektgruppe war, eine mächtige und schnelle *Suchfunktion* zur Verfügung zu stellen. Hierbei galt es mehrere Kriterien zu erfüllen. Die Suche sollte auf die Art der verwalteten Daten (BibTeX-ähnliche Informationen über Dokumente, siehe Abschnitt 5 auf Seite 38) zugeschnitten sein. Experten benötigen die Möglichkeit, komplexe *Suchanfragen* zu stellen. Für Nutzer ohne Vorkenntnisse war eine einfache Suche vorgesehen. Die Anfragesprache sollte an einer in wissenschaftlichen Kreisen bekannten Syntax ausgerichtet sein, um bei der Zielgruppe eine möglichst steile Lernkurve – auch für anspruchvollste Anfragen – zu erreichen.

Die nachfolgenden Abschnitte beschreiben die Entscheidungen, die innerhalb des Projektzeitraumes gefällt wurden, deren Ausführung sowie die Gründe für die jeweilige Entscheidung. In Abschnitt 13.1 auf Seite 106 sind einige Ideen aufgeführt, die von der nachfolgenden Projektgruppe entwickelt werden könnten.

9.1. Semantik

Es gibt zwei verschiedene Ergebnistypen bei Suchanfragen, zum Einen die Suche nach *Publikationen* und zum Anderen nach *Autoren*. Beide Anfragetypen haben den gleichen Aufbau. Eine Anfrage nach Autoren kann ohne syntaktische Änderung für Publikationen übernommen werden, lediglich die Ergebnisliste ist von einem anderen Typ.

Beispiel 1 *Gegeben sei folgende Anfrage:* `\author[lastname]{Schwentick}`
Wird mit diesem Ausdruck nach einem Autor gesucht, gibt der Algorithmus wie erwartet alle Autoren zurück, die den Nachnamen „Schwentick“ haben.
Der gleiche Ausdruck ergibt bei der Suche nach Publikationen eine Liste aller Publikationen, die von einem Autoren verfasst wurden, der den Nachnamen „Schwentick“ trägt.

Es ist möglich, nach dem Erscheinungsjahr oder -zeitraum, der Kategorie von Publikationen, Attributen von Autoren oder Publikationen und vielem mehr zu suchen. Ferner kann die Angabe einer Anzahl von Ergebnissen pro Seite, die anzuzeigende Seite sowie die Sortierung der Ergebnisse eingestellt werden. Eine eingehende Einführung in die Nutzung aller Suchfunktionen von oCIOBS wird in [24] gegeben.

9.2. Technologie

Bei den zu verwendenden Technologien war durch die allgemeine Entscheidung für die Programmiersprache Java und die Verwendung von Hibernate (Abschnitt 3.2 auf Seite 19) nur in Detailfragen Spielraum.

Die im ersten Semester der Projektgruppe entwickelte Hibernate-*Criteria*-Version der Suche stellte sich bei dem Test mit den Lehrstuhl-Daten als äußerst inperformant heraus. Knapp 500 Publikationen umfasste die Test-Datenbank. Komplexe Suchanfragen bewegten sich hier bereits bei mehr als einer halben Minute. Neben dem Overhead für das Mapping der objektorientierten Ausdrücke der Criteria API in SQL, kosten die SQL `JOIN`s, die für Unterabfragen erzeugt werden, Performance. Es bestand die Möglichkeit, spezielle Criteria-Sub-Queries zu verwenden, die in der Version 3.2 von Hibernate hinzukamen. Da einerseits nicht sicher war, ob die nötige Performanz mit diesem generellen objektrelationalen Ansatz erreicht und andererseits auf diese Weise keine Erweiterung der Datenbank für Volltextsuche eingebunden werden konnte, hat sich die Gruppe für einen Mittelweg entschieden. Parallel zu der Criteria-Variante ist in dem zweiten Semester der Projektgruppe eine auf nativem Standard-SQL basierende Suche entwickelt worden. Per *Properties* kann bei Verwendung der Datenbanksoftware *PostgreSQL* das Volltextsuche-Plugin *Tsearch2* (siehe [6]) verwendet werden. Die Criteria-Komponente der Suche wurde nicht mehr weiter gepflegt, da sie zu langsam war. Mit den SQL-Anfragen werden die `ID`s der gefundenen Entitäten zurückgegeben. Mit Hibernate werden dann die Objekte aus der Datenbank geholt. Dabei können die Geschwindigkeit einer optimierten SQL-Anfrage mit den Vorteilen des Hibernate-Cachings sowie dem komfortablen Mapping von relationen Tabellen auf Objekte von Hibernate, kombiniert werden. Die SQL-Suche überzeugt selbst bei den vollständigen Daten der DBLP mit ungefähr 800.000 Publikationen mit kurzen Suchzeiten von beinahe Nullzeit für einfache Anfragen, bis zu 10-15 Sekunden für sehr komplexe Suchanfragen mit vielen Ergebnissen.

Als Anfragesprache wurde eine LaTeX-ähnliche Syntax gewählt, da diese den meisten Wissenschaftlern – nicht nur Informatikern – geläufig ist. Für das Parsen der Anfragen hat sich die Gruppe für den Einsatz eines mit Hilfe eines *Compiler-Compilers* generierten *Parsers* entschieden, da dieser einfacher zu warten und besser strukturiert ist als eine manuelle Implementierung. Die Entscheidung, keinen eigens entwickelten Suchalgorithmus zu verwenden, der eine flexiblere und mächtigere Suche ermöglichen sollte, fiel gegen Mitte des ersten Projektsemesters. Die Verwendung von Hibernate erschwert eine zweistufige Realisierung, die eine Vorauswahl auf der Datenbank trifft und diese dann in Java weiter filtert. Es entsteht ein Overhead an Speicher und Laufzeit, der sich gerade dann negativ auswirken würde. Es gibt eine Bibliothek namens *Lucene*, die eine Volltextsuche auf Java-Basis implementiert. Eine Implementierung mit Hilfe dieser Technologie könnte die nachfolgende Projektgruppe vornehmen.

Des Weiteren hat die Gruppe festgestellt, dass Funktionen wie beispielsweise *re-*

guläre Ausdrücke zu mächtig für eine Suchanfrage sind, so dass die Nutzer derartig komplizierte Suchanfragen nicht verwendet hätten. Ferner lassen sich vereinfachte Funktionen mit Hibernate – beziehungsweise den dahinterliegenden *DBMS*-Systemen – abbilden. Hibernate bildet die relationalen Datenbanken auf Objekte ab. Dieser Vorgang ist nur mit speziell angelegten Tabellen, welche die Klassenstruktur abbilden, möglich. Diese Abstraktionsschicht vereinfacht den Zugriff auf die Datenbank, schränkt den Entwickler jedoch ein. Eine Optimierung der Datenbankabfragen ist – ohne Hibernate zu umgehen – ebensowenig möglich wie ein optimierter Indexierungsalgorithmus auf Java-Basis, der die Komplexität der Datenbankzugriffe bei Suchanfragen reduziert hätte. Der eingeschlagene Weg, die Suchanfragen mit SQL-Abfragen zu realisieren und die Objekte mit Hibernate zu holen erschien der Gruppe als bester Weg.

Die allgemeine Verfahrensweise, wie ein Suchformular in der Webclientoberfläche angezeigt und verarbeitet wird und wie die Kommunikation zwischen den Servlets und den *Search-Session-Beans* funktioniert, ist in Abschnitt 8 auf Seite 60 beschrieben. Der Zugriff durch einen Web Service auf die Suchfunktionen wird in Abschnitt 11 auf Seite 90 näher erläutert.

9.2.1. Parser

Suchanfragen werden entweder über die Oberfläche des Webclients oder über den Web Service an den *Suchalgorithmus* gesendet. Nach einem Vorverarbeitungsschritt, der die Anfrage sortiert, liest der `QueryParser` die Anfage ein und legt den zugehörigen *Syntaxbaum* im Speicher ab. Das weitere Verfahren ist in den Abschnitten 9.2.2 auf Seite 80 und 9.2.3 auf Seite 81 beschrieben.

Der Parser wurde mit *JavaCC* (siehe [11]) entwickelt. Bei dieser Technologie handelt es sich um einen *Parsergenerator* und *Lexer*, mit dem Parser für *LL(1)-Grammatiken* geschrieben werden können. Diese sind schwächer als *LR(1)-Grammatiken* und *LALR(1)-Grammatiken*. LL-Parser haben den Vorteil, dass sie sehr effizient sind, da es reicht, ein *Token* nach vorne zu sehen, um sich für eine *Produktion* zu entscheiden (siehe [9]). Die größte Einschränkung ist, dass *Linksrekursionen* nicht möglich sind.

Das Tool *JJTree* ermöglicht die Generierung von Syntaxbäumen. Die der Anfragesprache zugrunde liegende Grammatik ist in den Abbildungen 9.1 auf der nächsten Seite und 9.2 auf Seite 80 dargestellt. Die *Semantik* der Anfragesprache wird im Folgenden an einigen Beispielen verdeutlicht.

Beispiel 2 *Suche nach Publikation:*

```
\author[firstname]{Ingo} \author[lastname]{Wegener}
```

Ergebnis: Alle Publikationen von Autoren mit Vornamen „Ingo“ und Nachnamen „Wegener“.

```
BPAR: "{"
BOPTPAR: "["
PUBLICATION: "\publication"
CATEGORY: "\category"
CATEGORYLIST: "\categorylist"
AUTHOR: "\author"
PERIOD: "\period"
SEARCHIN: "\searchin"
EPAR: "}"
AND: "\and"
OR: "\or"
NOT: "\not"
OBRACE: "("
CBRACE: ")"
QUOTE: "\""
EOPTPAR: "]"
COMMA: ","
TRUE: "true"
FALSE: "false"
YEAR: (["0"-"9"]){4}
STRING: (~[ "{", "}", ",", "[", "]", "\\" ])+
```

Abbildung 9.1.: Terminale der Anfragesprache

Beispiel 3 *Suche nach Publikation:*

```
\author[homepage]{*ls5*} \author{Ralf Nagel}
```

Ergebnis: Alle Publikationen von Autoren mit einer Homepage, deren URL die Zeichenkette „ls5“ enthält und die als Namen eine der vier möglichen Kombinationen – „Ralf Nagel“, „Nagel Ralf“, „Ralf Ralf“, „Nagel Nagel“ – von Vor- und Nachname haben.

Beispiel 4 *Suche nach Autoren:*

```
\year{1945,1999}
```

Ergebnis: Alle Autoren die zwischen „1945“ und „1999“ publiziert haben.

Die Anfragesprache enthält keine regulären Ausdrücke, da einerseits nicht alle DBMS, die Hibernate unterstützt, diese Funktionalität aufweisen und *rechtslineare Grammatiken* andererseits für den durchschnittlichen Nutzer zu kompliziert sind. Des Weiteren bildet der Like-Operator von SQL beliebige Zeichen und Zeichenketten ab (in der Anfragesprache dargestellt durch „?“ und „*“), welche im Normalfall ausreichen.

```
query := (( <PUBLICATION> ((optArg arg) | arg)) |
    (<AUTHOR> ((optArg arg) | arg)) |
    (<CATEGORY>arg) |
    (<CATEGORYLIST>categoryListArg) |
    (<PERIOD>periodArg) |
    exp)+
arg := <BPAR> exp <EPAR>
optArg := <BOPTPAR> list <EOPTPAR>
periodArg := <BPAR> (<YEAR>)? <COMMA> (<YEAR>)? <EPAR>
categoryListArg := <BPAR> list <EPAR>
booleanArg := <BPAR> ((<TRUE>) | (<FALSE>)) <EPAR>
exp := andexp (<OR> andexp)*
andexp := term (<AND> term)*
term := (value | (<NOT> (value | groupexp)) | (<QUOTE>
         value <QUOTE> ) | groupexp )
groupexp := <OBRACE> exp <CBRACE>
value := <STRING>
list := (<STRING> (<COMMA>)?)+
```

Abbildung 9.2.: Nicht-Terminale der Anfragesprache

9.2.2. Syntaxbaum und Visitor

Syntaxbaum Der Query-Parser erzeugt einen Syntaxbaum mit getypten Knoten, welcher im darauf folgenden Schritt traversiert wird. Ein Beispiel ist in Abbildung 9.3 auf der nächsten Seite dargestellt.

Visitor Für einen Durchlauf des Baumes wird das sogenannte *Visitor-Konzept* verwendet, welches von JJTree unterstützt wird. Dabei wird zuerst die Wurzel angestoßen. Jeder angestoßene Knoten akzeptiert den Besucher, indem er auf der Visitor-Klasse die `visit()`-Methode aufruft. Diese Methode ist für jeden Knotentyp überladen. Das ermöglicht eine gesonderte Behandlung von unterschiedlichen Knotentypen. In der `visit()`-Methode wird der Knoten verarbeitet. Handelt es sich um einen inneren Knoten wird im Normalfall die `acceptChildren()`-Methode auf dem aktuellen Knoten aufgerufen. Diese stößt alle Kindknoten an. Es handelt sich demnach um eine *Breitensuche* (BFS). Ausnahmen bilden Knoten, die fest definierte Kindknoten haben. Diese werden direkt in dem Elterknoten bearbeitet, ohne selbst angestoßen zu werden. In den Abschnitten „Criteria-Komponente“ auf der nächsten Seite und „SQL-Komponente“ auf der nächsten Seite wird näher erläutert, wie in der Visitor-Klasse die Suchanfrage erstellt wird.

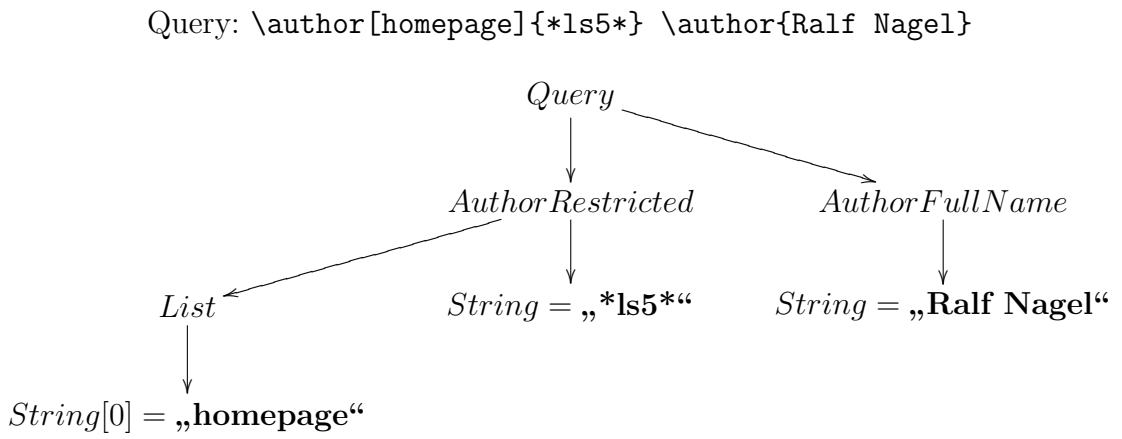


Abbildung 9.3.: Beispiel-Syntaxbaum

9.2.3. Criteria-Komponente

Für die Anfrage an die Datenbank wurde im ersten Semester des Projektes die Criteria-API verwendet (siehe Abschnitt 3.2.4 auf Seite 20). Im zweiten Semester ist eine zweite Suchkomponente, die SQL erzeugt, entwickelt worden (siehe Abschnitt „SQL-Komponente“ auf dieser Seite). Die `visit()`-Methoden setzen den Criteria-Ausdruck zusammen. Ferner fügt der `Visitor` noch einige Parameter, wie zum Beispiel die Sortierung und die Anzahl von Einträgen pro Seite, ein. Das Ergebnis der Datenbankanfrage wird in einer Liste von Autoren- beziehungsweise Publikationsobjekten an den Web Service oder die Webclientoberfläche zurückgegeben.

9.2.4. SQL-Komponente

Zu Beginn des zweiten Semester der Projektgruppe standen erstmals Daten zu konkreten Autoren und Publikationen zum Testen der Suche zur Verfügung. Dabei handelte es sich um die Veröffentlichungen am Lehrstuhl 5. Die Suche hatte mit der in Abschnitt „Criteria-Komponente“ auf dieser Seite beschriebenen Suchkomponente bei ungefähr 500 Publikationen Suchzeiten von mehr als einer halben Minute. Daher entschied sich die Gruppe, eine zweite Komponente zu entwickeln, die SQL erzeugt, welches standardkompatibel ist. In Property-Dateien können datenbankspezifische Eigenschaften hinzugeschaltet werden. Bisher kann die Volltextsuche Tsearch2 von PostgreSQL verwendet werden. Weitere Besonderheiten von Datenbanken könnten von der Nachfolgeprojektwoche entwickelt werden. Näheres kann dazu dem „Ausblick für die Suchfunktion“ auf Seite 106 entnommen werden.

Die SQL-Suche traversiert ähnlich wie die Criteria-Komponente den Syntaxbaum zu der Suchanfrage mit Hilfe eines Visitors und konstruiert dabei den SQL-String. Im Gegensatz zu der Criteria-Variante liefert die Datenbankabfrage die `ID`s der En-

titäten zurück. In einem zweiten Schritt werden die Entitäten mit Hibernate abgerufen.

9.3. Varianten

Die Suche wird in verschiedenen Varianten angeboten. Der Webclient bietet eine Expertensuche (Abschnitt 9.3.1), die auch als Schnellsuche verwendet werden kann sowie eine erweiterte Suche (Abschnitt 9.3.2). Weiterhin gibt es einen Web-Service-Adapter für die Suchfunktion, der beispielsweise von der Administrationsoberfläche eingesetzt wird. Der Web Service soll eine plattformunabhängige Schnittstelle zu der Suche zur Verfügung stellen. Dies ermöglicht die Einbindung dieser Funktionen in beliebige (Fremd-) Anwendungen.

9.3.1. Expertensuche

Die Expertensuche stellt im Webclienten ein schlichtes Webformular mit einem Textfeld, einem Sendeknopf sowie einem Auswahlknopf, der festlegt, ob nach Autoren oder Publikationen gesucht werden soll (siehe Abbildung 9.4), zur Verfügung. Ferner kann in einer Auswahlbox die Anzahl der Suchergebnisse pro Seite gewählt werden.

In das Textfeld kann ein komplexer Suchausdruck eingegeben werden. Für die einfache Suche besteht die Möglichkeit, sich auf einen Ausdruck auf der Ebene der Expressions (siehe Abbildung 9.2 auf Seite 80) zu beschränken. Bei der Suche nach Autoren entspricht das dem Aufruf von `\author{exp}` und bei der Suche nach Publikationen dem Ausdruck `\publication{exp}`. Nähere Informationen zur Benutzung der Suche sind auf der oCOBS-MediaWiki-Seite [24] zu finden.

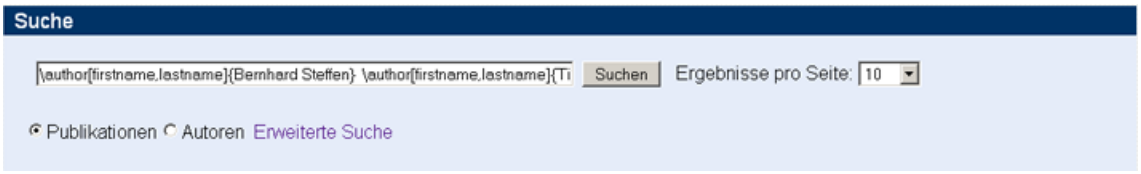


Abbildung 9.4.: Suchmaske der Experten- und Schnellsuche.

9.3.2. Erweiterte Suche

Die erweiterte Suche (siehe Abbildung 9.5 auf der nächsten Seite) bietet einzelne Suchfelder für die Vor- und Nachnamen von bis zu drei Autoren, den Titel einer Publikation, die Kategorie, zu welcher die Publikation gehört sowie zwei Felder für die Angabe des Veröffentlichungzeitraums. Ferner ist es möglich, wie im Falle der einfachen Suche, zwischen der Suche nach Publikationen und der Suche nach Autoren zu

Abbildung 9.5.: Suchmaske der erweiterten Suche.

wählen. Die aufgelisteten Felder vereinfachen die Eingabe. Sie sind für eine einfache Suche wiederum zu umfangreich. Weiterhin lassen sich nicht alle Sprachkonstrukte auf diese Felder abbilden. Daraus resultiert die Daseinsberechtigung für alle Formen der Suche.

Die Werte der einzelnen Felder werden in dem verarbeitenden Servlet in eine Suchanfrage, die der Parser versteht, übersetzt, bevor sie an das Search-Session-Bean gesendet werden.

Beispiel 5 *In dem Suchformular seien die Felder Titel, Vor- und Nachname des Autors ausgefüllt. Der resultierende Ausdruck sieht wie folgt aus:*

```
\author[title]{Prof. Dr.} \
\author[firstname]{Bernhard} \
\author[lastname]{Steffen}
```

Die „Backslash“-Zeichen am Ende der ersten beiden Zeilen bedeuten, dass der Ausdruck in der nächsten Zeile weitergeht.

9.4. Testen der Suche

Die Wahl der Technologien (siehe Abschnitt „Technologie“ auf Seite 77) erschwerte die Verwendung von Standardtestverfahren wie *Unit-Tests*. Große Teile des Codes sind aus der Beschreibung des `QueryParser`s generierter Quellcode. Ferner benötigen die `visit()`-Methoden des Visitors einen JTree beziehungsweise einen Knoten, der zu einem validen Syntaxbaum eines Anfrageausdrucks gehört. Ein erheblicher Aufwand wäre für die Initialisierung eines jeden Tests die Folge gewesen. Weitere Anforderung an die Testmethoden der Suche waren die Abdeckung der Einstellungsmöglichkeiten und der Komponenten der Suchsprache. Die Umstellung von

Hibernate Criteria auf Standard SQL sollte keine Änderung der Suchsemantik verursachen. Dies erfordert die Möglichkeit, einen Vergleich der Implementierungen durchzuführen.

Es wurde ein eigenes Testverfahren entwickelt, das Gebrauch von dem *JUnit-Framework* macht, um die oben genannten Kriterien zu erfüllen. Die Suche wird als „Black Box“ behandelt. Durch Vergleich mit Referenzwerten gewährleisten die Tests die Konsistenz der Anwendung über den Entwicklungszeitraum:

Initial iteriert der Test in einer Schleife über Suchanfragen, die möglichst viele Kombinationen von Komponenten der Anfragesprache abdecken. Jede Anfrage wird für alle Einstellungskombinationen (zum Beispiel Sortierung, Suche nach Autoren oder Publikationen) ausgeführt. Die Laufzeitumgebung verwendet weder den Webclient noch den Applikationsserver. Die Ergebnisse der Suchanfragen werden in eine *XML-Datei* serialisiert. Bei erwünschten Fehlerfällen wie Syntaxfehlern in der Suchanfrage folgt eine Speicherung der Ausnahme. Valide Referenzwerte erfordern die einmalige Prüfung der Ausgaben. Bei Änderungen in der Implementierung können die gespeicherten Anfragen erneut ausgeführt und Abweichungen festgestellt werden. Differenzen können aus Fehlern in beiden Implementierungen resultieren. Diese Ergebnisse sind erneut zu überprüfen.

9.5. Probleme bei der Entwicklung

Es erwies sich als schwierig, eine in Java programmierte Vorauswahl oder einen anschließenden Filter in den Suchalgorithmus einzubinden. Da Hibernate (siehe Abschnitt 3.2) einerseits die Anbindung an die SQL-Datenbank und andererseits das Mapping von relationalen Datenbanktabellen auf Objekte übernimmt, sind Optimierungen nur sehr eingeschränkt möglich. Es besteht die Möglichkeit, Hibernate zu umgehen, doch diese Lösung ist zum Einen unsauber und zum Anderen wird auch die Tabellenstruktur auf die objekt-relationale Abbildung zugeschnitten. Das Mapping zwischen Objekten und Tabellen hat den weiteren Nachteil, dass Anfragen per HQL oder Criteria andere Namen verwenden als die Datenbank. Ferner sind Anfragen mit HQL und Criteria durch die Abstraktion von der jeweiligen Datenbank und den Overhead von Hibernate zumeist viel langsamer als die Verwendung von SQL-Anfragen.

10. Adminbackend

Von Anfang an wurde von der Projektgruppe die Idee gefördert, weitestgehend unabhängig von individuellen Technologien[1] zu sein. Im Falle der Administrationsoberfläche führte dies zu der Entscheidung, eine Zwischenschicht hineinzuziehen, die den Datenbestand mit der GUI verbindet. So wird die Möglichkeit offengehalten, in Zukunft den HTML-basierten Adminclient (siehe Abschnitt 11 auf Seite 90) durch ein browserunabhängiges Programm zu ersetzen. Um darüber hinaus auch noch eine gewisse Plattform- und Programmiersprachenunabhängigkeit zu erreichen, entschied die Gruppe, diese Kommunikationsschicht mittels Web Services zu realisieren.

Das aus dieser Anforderung resultierende Adminbackend stellt demnach also im Wesentlichen Web Services zur Verfügung, mit denen der Datenbestand des oCIOBS gelesen und verändert werden kann, sodass ein beliebiger Adminclient (also auch die HTML-Implementierung der Projektgruppe) alle Methoden zur Verfügung hat, die er benötigt. Insbesondere können diese Funktionen über das Netzwerk aufgerufen werden, sodass der Adminclient auf beliebigen Rechnern laufen kann, die den Backend-Rechner erreichen können.

10.1. Allgemeiner Aufbau

Das Adminbackend besteht aus einer recht geringen Anzahl an Klassen, von denen die meisten allerdings pro angebotenem Service eine Methode besitzen. Dabei sind die Funktionen für sich betrachtet sehr allgemein gehalten und bieten jeweils kleine Einzelfunktionalitäten an. So ist der Adminclient nicht an einen vom Adminbackend festgelegten Bildschirmaufbau gebunden, muss aber unter Umständen pro Bildschirm mehrere Funktionen des Backends aufrufen. Diese Steigerung der Flexibilität resultiert insgesamt in einer hohen Anzahl relativ kleiner Funktionen.

Abbildung 10.1 auf der nächsten Seite verdeutlicht die Interaktion zwischen den Klassen: Das Interface `AdminInterface` beschreibt die Signaturen aller angebotenen Methoden und legt mittels Annotation fest, welches `AtomicRight` benötigt wird, um diese auszuführen. Der `ExecutionDispatcher` wird vom Web Service aufgerufen, prüft die Rechte und ruft die entsprechende Methode aus der `AdminBean` auf, die dann letztendlich die gewünschte Funktion implementiert.

[1] Java EE war als Basistechnologie durch die Aufgabenstellung der PG gefordert, nicht aber die genauen Implementierungen dieser Spezifikationen.

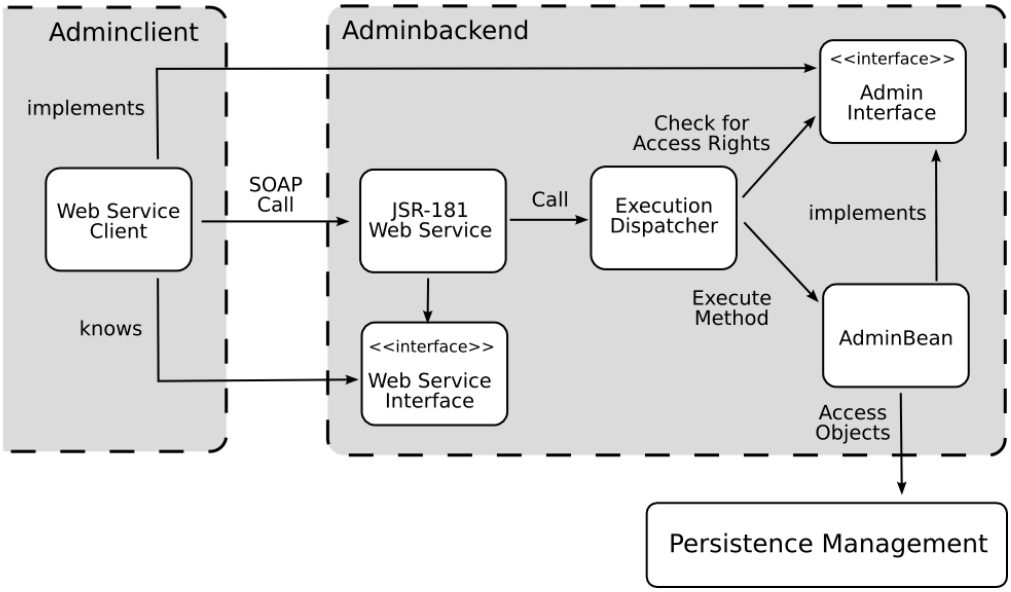


Abbildung 10.1.: Aufbau des Adminbackends

10.2. Zugriffskontrolle

Naturgemäß müssen die Zugriffe auf die administrativen Funktionen gesichert werden, damit Unbefugte keinen Zugriff darauf haben. Aus diesem Grund gibt es `User`, die sich mit einem Passwort am System anmelden müssen. Wenn die Kombination Benutzername/Passwort korrekt ist, bekommt der Anwender eine Session-ID zugewiesen, die beim darauf folgenden Aufrufen von Methoden immer übergeben werden muss, um die Session (und damit den Anwender) zu identifizieren.

Da die Sessions eine geringe Lebensdauer haben, lohnt es sich nicht, diese über die Persistenzschicht (und damit die Datenbank) zu verwalten. Um die Sessions aber dennoch in unterschiedlichen *Stateless SessionBeans* verwenden zu können, werden diese als Map, die von der Session-ID auf die Session abbildet, im *JNDI*-Baum abgelegt.

Es wurde bewusst auf die Verwendung der mitgelieferten HTTP-Session-Verwaltung des Application Servers verzichtet, damit die Unabhängigkeit von Technologien gewahrt bleibt. So wäre bei der Verwendung von HTTP-Sessions kein Client mehr möglich gewesen, der nicht auf HTTP aufsetzt.

Ein Nutzer kann viele Rollen haben, welchen wiederum viele Zugriffrechte zugewiesen sein können. Durch Vereinigung aller Rechte aller Rollen eines Nutzers ergibt sich die gesamte Menge an Rechten, die ein Benutzer hat.

Wie bereits im vorherigen Abschnitt angedeutet, enthält jede Methode des `AdminInterface`s eine Annotation des Typs `@AccessControl`, die ihr ein (*genau*

ein) `AtomicRight` zuweist. Ein Benutzer darf demnach alle Methoden ausführen, die ein annotiertes `AtomicRight` haben, welches einer seiner `Role`s zugewiesen ist.

10.3. Implementierung

Die Klasse `AdminBean` ist eine Implementierung des `AdminInterface` und enthält den eigentlichen Code, der auf die Persistenzschicht (siehe Kapitel 6 auf Seite 46) zugreift, um damit den Datenbestand zu manipulieren. Die vielen kleinen Methoden werden recht ähnlich implementiert, weswegen hier nur das allgemeine Vorgehen erläutert und an einem Beispiel verdeutlicht werden soll. Die Implementierung einer der Methoden besteht dann meist aus wenigen Zeilen Code (siehe Abbildung 10.2).

```
public void publicationSetTitle(long publicationId,
String title) throws IllegalParameterException {

  // Load Publication from database
  Publication publication =
    getElementFromPM(Publication.class, publicationId);
  // Update title
  publication.setTitle(title);

  // Create new LogEntry
  LogEntryPublication lep = new LogEntryPublication(
      getAtomicRight(), null, getUser(),
      "Title changed to " + title, publication);
  // Save LogEntry to database
  writeElementToPM(lep);
}
```

Abbildung 10.2.: Eine Beispielmethode aus der AdminBean

Da die Objekte nicht direkt manipuliert werden können, sondern zunächst für die Übertragung serialisiert werden müssen, kann man an die Methoden des Adminbackends nicht die zu manipulierenden Objekte selbst übergeben, sondern nur die ID, durch welche das Objekt innerhalb des Persistenzmechanismus eindeutig identifiziert wird. Anhand dieser ID wird dann das zu bearbeitende Objekt mit Hilfe der Persistenzschicht geladen. Entsprechend der weiteren Methodenparameter wird dann dieses Objekt verändert. Die Behandlung der Transaktionen wird global im `ExecutionDispatcher` behandelt, sodass an dieser Stelle kein commit erfolgen muss.

Methoden, die neue Objekte erstellen (wie zum Beispiel `createPublication()`), haben in der Regel als Rückgabewert die ID des neu erstellten Objektes in der

Datenbank, damit der Adminclient das neue Objekt bei folgenden Aufrufen direkt referenzieren kann.

Zusätzlich zu der eigentlichen Änderung an den Daten wird auch in jeder Methode ein entsprechender (und spezialisierter, siehe Kapitel Datenmodell auf Seite 38) `LogEntry` erzeugt, der die ausgeführte Aktion menschenlesbar protokolliert. Dies ermöglicht zwar kein automatisches Rückgängigmachen, aber ein etwaiger Missbrauch des Administrationsdienstes kann von einem Menschen verfolgt und untersucht werden.

10.4. Web Services

Die grundlegende Schwierigkeit beim Anbinden des Adminclients an das Adminbackend mittels Web Services bestand darin, eine geeignete Möglichkeit zum Serialisieren der Daten für den Lesezugriff zu finden. Eine automatische, durch das Web-Service-Framework zur Verfügung gestellte Serialisierung kam im Wesentlichen aus zwei Gründen nicht in Frage: Zum einen funktioniert sie nur bei JavaBeans mit primitiven Datentypen[2], zum anderen ist die oCOBS-Datenstruktur so stark zusammenhängend, dass im ungünstigsten Fall mit jedem Aufruf der gesamte Datenbestand serialisiert und versendet worden wäre.

Aus diesem Grund entwickelte die Projektgruppe ein Konzept, in dem auf Seite des Adminclients Proxy-Objekte erstellt werden, die – da sie von den jeweiligen Datenbasis-Klassen erben – genauso wie die *echten* Objekte aussehen. Allerdings enthalten sie selbst keine Assoziazionen, sondern lediglich die entsprechenden IDs der referenzierten Objekte. Wird nun auf einem Proxy hierfür ein Getter aufgerufen, dann ruft der Proxy selbst den dazugehörigen Web Service auf, der das Objekt zurückgibt. Das vom Proxy zurückgegebene Objekt ist dann meist wieder ein Proxy.

Damit steht dem Adminfrontend für den lesenden Zugriff die volle API der Datenbasis zur Verfügung, und kann trotzdem entfernt angebunden werden. Mit Ausnahme der erhöhten Latenz gibt es keinen Unterschied zum Arbeiten mit den echten Datenbankobjekten. Die Überlegung, auch den schreibenden Zugriff über die Proxy-Objekte zu behandeln, kam zwischenzeitlich auf, wurde aber verworfen, da damit das Rechtemodell mit den annotierten `AtomicRight`s so nicht hätte beibehalten werden können.

Für den Transport (und damit für die Signatur des Web Services) wurden separate Klassen erstellt, die ausschließlich primitive Typen als Felder enthalten. Kompliziertere Assoziationen werden hierbei in (zum Teil mehrdimensionale) Arrays übersetzt. Die Transportklassen können sich selbst aus einem Datenbankobjekt initialisieren. Analog kann an die Proxies jeweils auch ein entsprechendes Transportobjekt zur Selbstinitialisierung übergeben werden.

[2]Schon bei den `Collection`-Interfaces versagte die automatische Serialisierung des JBossWS.

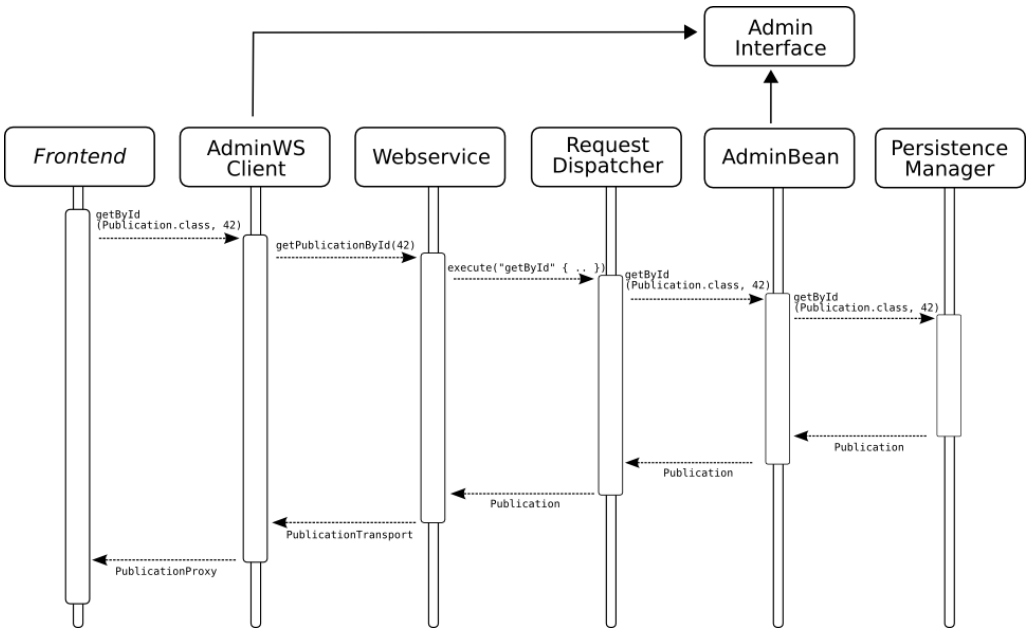


Abbildung 10.3.: Lesezugriff mittels Transport- und Proxy-Klassen

Abbildung 10.3 zeigt in einem Sequenzdiagramm die Abfolge der Aufrufe um eine `Publication` anhand ihrer Datenbank-ID zu laden. Dort ist zu sehen, dass bis zum Webservice die Rückgabe das echte `Publication`-Objekt ist. Dort wird dieses dann in ein `PublicationTransport` umgewandelt und auf Seiten des AdminWSClient als `PublicationProxy` an den eigentlichen Aufrufer zurückgegeben.

11. Adminclient

Dieses Kapitel beschreibt die Administrationsschnittstelle. Zuerst wird auf die Anforderungen an die Schnittstelle eingegangen. Danach folgt die Beschreibung der technischen Realisierung. Anschließend wird ein Einblick in die entworfene Oberfläche gegeben. Die angebotenen Funktionen und Workflows werden mit Hilfe der einzelnen Seiten erläutert.

11.1. Anforderungen an die Administrationsschnittstelle

Die Administrationsschnittstelle hat gewissen Anforderungen zu genügen. Ein Teil der Anforderungen ist durch die Projektbeschreibung gegeben. Zu erwähnen sind:

- Die Wartung und Pflege eines Datenbestandes, der höchst inhomogen ist,
- eine Benutzerverwaltung, die dem komplexen, mehrstufigen Prozess der Datenpflege Rechnung trägt,
- Einfachheit der Verwaltung, und
- Möglichkeiten, den Betrieb von oCIOBS zu überwachen.

Zusätzlich zu diesen Anforderungen hat sich die Projektgruppe entschieden, eine Architektur zu wählen, welche leicht erweiterbar und flexibel ist. Darüber hinaus sollten Sicherheitsaspekte, zu nennen sind hier besonders Authentifizierung und Integrität, adäquat unterstützt werden. Weiterhin sollte die gewählte Technologie den Benutzer darin unterstützen, die Konsistenz der Daten zu bewahren. Insbesondere sollten eingegebene Daten einfach zu validieren sein. Die vorgeschriebenen Workflows sollten durch eine funktionale Oberfläche unterstützt werden.

11.2. Technische Umsetzung

Dieses Kapitel beschäftigt sich mit den von der Projektgruppe erarbeiteten Lösungen. Hierbei wird auf Designentscheidungen eingegangen, wie auch auf deren technische Umsetzung.

11.2.1. Web Application

Um der Anforderung nach einer einfachen und flexiblen Lösung nachzukommen, hat sich die Gruppe dazu entschlossen, die Administrationsoberfläche als Web Application zu realisieren. Als zusätzlicher Vorteil dieser Lösung wurde weiterhin die Arbeitsplatzunabhängigkeit und daraus resultierte Flexibilität gesehen, die gegenüber einer herkömmlichen Anwendung gegeben ist. Zur Realisierung der Weboberfläche wurde *JavaServer Faces* gewählt[1]. Diese Technologie bietet komfortable Möglichkeiten zur Validierung von Eingaben an. Weiterhin ist es möglich, einzelne Elemente der Oberfläche dynamisch zu erzeugen und zu ändern.

Die Administrationsoberfläche des oCIOBS ist aufgrund hoher Anforderungen an Benutzerfreundlichkeit und Erweiterbarkeit weitgehend mit aktuellen Techniken und Standards wie *JSF 1.1*, *CSS 2.1* und *XHTML 1.0*, umgesetzt worden. Die Nutzung aktueller Browser für den fehlerfreien Einsatz des Systems ist zu empfehlen. Diese Einschränkung ist für den Adminclient zu rechtfertigen, da er – im Gegensatz zum Webclienten – nur von einem ausgewählten Kreis genutzt wird.

11.2.2. Web Services

Die Flexibilität der Lösung wird durch die Verwendung von Web Services, auf denen die Administrationsschnittstelle aufsetzt, noch weiter erhöht. So greift die Administrationsoberfläche niemals direkt auf die oCIOBS-Anwendung zu, sondern nutzt Web Services, welche diese über das Admin-Backend zu Verfügung stellt. Für genauere Details zur Web-Service-Schnittstelle des oCIOBS, siehe Kapitel 10 auf Seite 85.

11.3. Überblick über das System

Wie bereits erwähnt, ist die Administrationsschnittstelle des oCIOBS ein webbasiertes System. Die Hauptaufgabe dieses Systems besteht in der Aktualisierung und Erweiterung der Datenbank. Die bisher in der DBLP eingesetzten Methoden zur Aktualisierung und Pflege der Daten sind sehr umständlich und meist manueller Natur. Deshalb wird bei der Neuentwicklung der DBLP-Datenbank großer Wert darauf gelegt, die Schwachstellen der bisherigen Methoden auszumerzen, um ein möglichst einfach und intuitiv bedienbares System zu schaffen. Dabei werden folgende Verbesserungen angestrebt:

- Ermöglichen einfachen und intuitiven Arbeitens mit dem System,
- Erschaffung eines rollenbasierten Systems für eine bessere und einfachere Verteilung der Aufgaben und mehr Sicherheit,

[1] Einzelheiten zur Entscheidungsfindung sowie zur Technologie sind im Abschnitt 3.5 auf Seite 24 zu finden

- mehr Transparenz in Bezug auf die geleisteten und zu leistenden Aufgaben bei der Aktualisierung,
- Protokollierung der Ereignisse und
- durchgehende und konsistente Workflows.

In den folgenden Unterkapiteln wird die Navigationsstruktur des Systems aufgezeigt, um einen Überblick über den Aufbau und die Strukturierung der Oberfläche zu erhalten. Danach werden die Workflows vorgestellt um die Gemeinsamkeiten dieser hervorzuheben. Anschließend wird auf die einzelnen Bereiche des Systems genauer eingegangen.

11.3.1. Aufbau und Navigation

Die einzelnen Seiten der Administrationsoberfläche werden grob in drei Bereiche unterteilt:

1. Kopfbereich,
2. Navigationsbereich und
3. Inhaltsbereich.

Abbildung 11.1.: oCOBS Administrationsoberfläche im Überblick

Abbildung 11.1 zeigt einen Screenshot in dem die drei Bereiche farblich markiert sind.

Der Kopfbereich

Dieser Bereich ist immer sichtbar. Die rechte Seite zeigt den Anmeldenamen des Benutzers. Rechts daneben befindet sich ein Link zur Abmeldung aus dem System. In diesem Bereich wird auch ein kleines Formular eingeblendet. Dieses Formular ermöglicht eine schnelle Suche und Weiterleitung zu einer bestimmten Publikation anhand ihrer ID.

Der Navigationsbereich

In diesem Bereich können ein bis drei Navigationsebenen eingeblendet sein. Diese sind farblich, in Schriftgröße und Form voneinander zu unterscheiden. Die erste Navigationsebene ist blau hinterlegt. Die zweite Navigationsebene ist grau hinterlegt, wohingegen die dritte und letzte in einer Reiter-Form dargestellt ist. Navigationspunkte und Ebenen können je nach Berechtigung des Benutzers und Zustand der Seite beziehungsweise des darzustellenden Objektes eingeblendet, ausgeblendet, aktiviert oder deaktiviert sein.

Die erste Navigationsebene ist immer sichtbar und dient der Strukturierung. Alle Hauptbereiche des Systems sind durch diese Navigation zu erreichen.

Mit Ausnahme der Startseite wird durch Auswahl einer dieser Menüpunkte eine zweite Navigationsebene eingeblendet. Durch diese Ebene erreicht man alle zu dieser Rubrik vorhandenen Funktionen. So kann ein Benutzer in der Rubrik *Publikationen* je nach Berechtigung beispielsweise

- nach Publikationen suchen,
- Publikationen anlegen, editieren und löschen,
- Attribute für die Publikationen anlegen, editieren und löschen sowie
- Publikationsvorlagen anlegen, editieren und löschen.

Anschließend werden die einzelnen Funktionen der zweiten Navigationsebene in weitere Optionen oder Schritte unterteilt.

In dem Abschnitt *Publikationen suchen*, hat ein Redakteur verschiedene Suchoptionen zur Auswahl. Er kann die bekannten Sucharten aus dem *Webclient*, also die Standardsuche und die erweiterte Suche verwenden (siehe hierzu auch 9 auf Seite 76). Weiterhin stehen bestimmten Benutzergruppen spezielle Suchfilter zur Verfügung, wie

- vom aktuellen Nutzer zuletzt bearbeiteten Publikationen,
- unvollständige Publikationen,
- vorgeschlagene Publikationen oder

- zurückgenommene Publikationen.

Diese Ebene kann auch die einzelnen voneinander unabhängigen Schritte einer Funktion darstellen. Das Anlegen einer neuen Publikation ist beispielsweise in die folgenden Schritte unterteilt:

- Publikation erzeugen,
- Publikationsfelder bearbeiten,
- Kategorien bearbeiten,
- Autoren bearbeiten, und
- Überprüfen, Freigeben und Publizieren.

11.3.2. Allgemeine Eigenschaften der Workflows

Wie dem Abschnitt 11.3.1 auf Seite 92 zu entnehmen ist, sind die einzelnen Bereiche des Adminclients sehr ähnlich aufgebaut. Dies erleichtert den Umgang mit dem System. Der Redakteur sollte nach einer kurzen Einführungszeit in der Lage sein, mit dem System umzugehen, ohne etwas über den Aufbau der Datenbank zu wissen. Die häufig benutzten Funktionen wie das Anlegen von Publikationen, Autoren oder Kategorien sind sich sehr ähnlich und übersichtlich.

11.4. Adminclient im Detail

Dieses Kapitel gibt einen Überblick über die Funktionen der Administrationsschnittstelle. Die folgenden Kapitel entsprechen den Hauptnavigationspunkten der Administrationsoberfläche.

11.4.1. Kategorien verwalten

In diesem Bereich kann der Benutzer nach Kategorien suchen, neue Kategorien anlegen sowie die Kategorienzuordnungen bearbeiten.

Kategorien durchsuchen: Der *Kategorie-Browser* ist an einen üblichen Dateibrowser angelehnt und soll es dem Benutzer ermöglichen, komfortabel durch die Kategorienstruktur zu navigieren. Dies geschieht entweder vom Root oder anhand eines Kürzels. Der Benutzer kann hier eine Ober- bzw. Unterkategorie der aktuell ausgewählten Kategorie auswählen und somit durch den Graph navigieren. Alle auf der Seite sichtbaren Kategorien können direkt bearbeitet werden, sofern der Benutzer dazu berechtigt ist.

Zusätzlich zu der aktuell ausgewählten Kategorie, werden alle Pfade zum Root angezeigt. Dies soll es einem Benutzer erleichtern, die Konsequenzen abzuschätzen, die das Zuordnen einer Unterkategorie zu einer Überkategorie hat. Mit Konsequenzen sind einerseits Zyklen gemeint, die durch Einfügen von Kategorien entstehen können. Andererseits muss überprüft werden können, ob die Pfade, die zu dieser Kategorie führen, auch in sich logisch sind oder vielleicht Überkategorien die Anwesenheit der zu bearbeitenden Kategorie ausschließen.

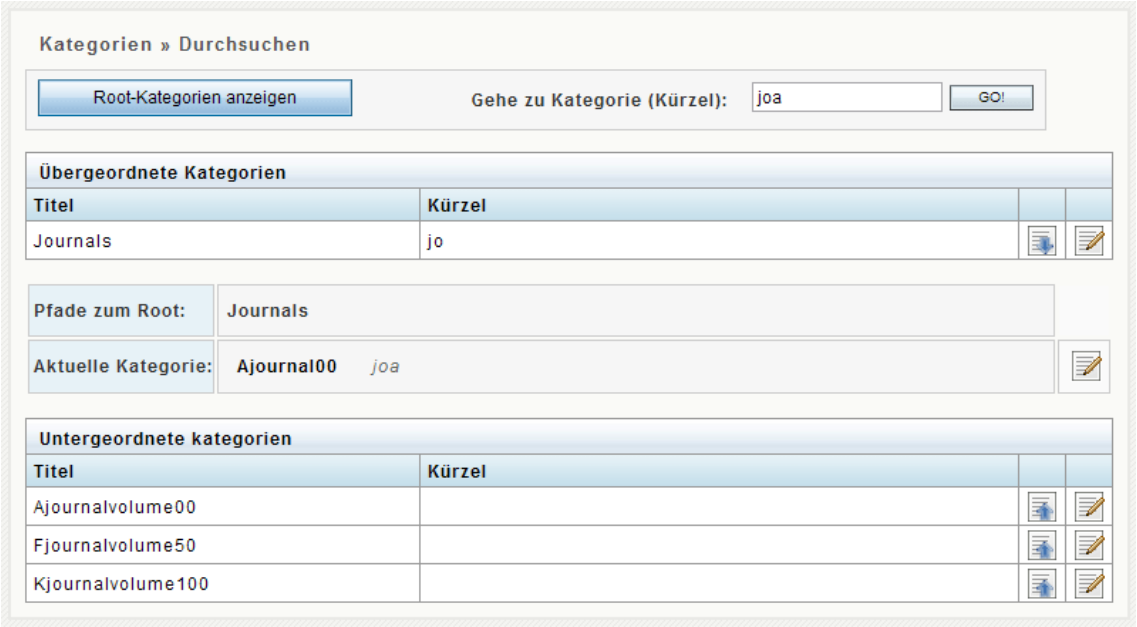


Abbildung 11.2.: Kategorie-Browser

Kategorie anlegen/bearbeiten: Das Anlegen einer neuen Kategorie erfolgt über die Angabe eines Kategorienamens, einer optionalen Beschreibung sowie einer Vorlage innerhalb des Menüpunktes *Eine neue Kategorie hinzufügen.* (siehe Abbildung 11.3 auf der nächsten Seite)

Die Bearbeitung erfolgt in derselben Maske. Jedoch kann die Vorlage nachträglich nicht editiert werden.

Bestehende Kategorien können mit Hilfe des Kategorie-Browsers in den bestehenden Kategoriengraph eingeordnet werden.

11.4.2. Publikationen verwalten

Die Publikationszone ist wahrscheinlich der wichtigste und im späteren Einsatz der meistverwendete Bereich innerhalb der Administrationsschnittstelle. In diesem Bereich kann man nach Publikationen suchen, neue Publikationen und Publikations-

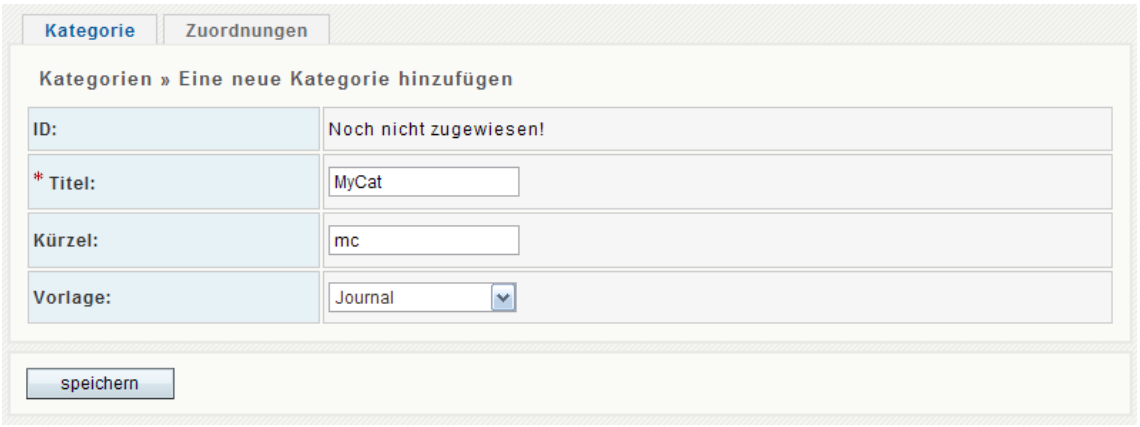


Abbildung 11.3.: Kategorie anlegen

attribute erfassen sowie bearbeiten und schließlich Vorlagen für die Publikationen erstellen und bearbeiten.

Publikationen suchen

Hier kann der Benutzer entweder die bekannte Suche (Siehe Kapitel 9 auf Seite 76) aus dem Web-Client verwenden oder eine der folgenden Suchabfragen starten:

- *Von mir zuletzt bearbeitete Publikationen*,
- *unvollständige Publikationen*,
- *vorgeschlagene Publikationen* und
- *zurückgenommene Publikationen*.

Neue Publikation hinzufügen

Das Erzeugen einer Publikation zerfällt in vier Bearbeitungsschritte und endet mit einer Übersicht, die der Überprüfung dient. Die Bearbeitungsschritte zur Erstellung einer Publikation beginnen mit dem Anlegen einer neuen Publikation. Manipulationen bestehender Publikationen können dann in beliebiger Reihenfolge erfolgen und jederzeit unterbrochen werden. In dem ersten Schritt *Publikation* wird eine neue Publikation angelegt und in der Datenbank gespeichert (siehe auch Abbildung 11.4). Dieser Schritt beinhaltet das Ausfüllen von drei Pflichtfeldern: Titel der Publikation, Veröffentlichungsjahr und die Auswahl einer Vorlage. Erst nachdem diese Felder ausgefüllt sind, können die weiteren Schritte durchgeführt werden.

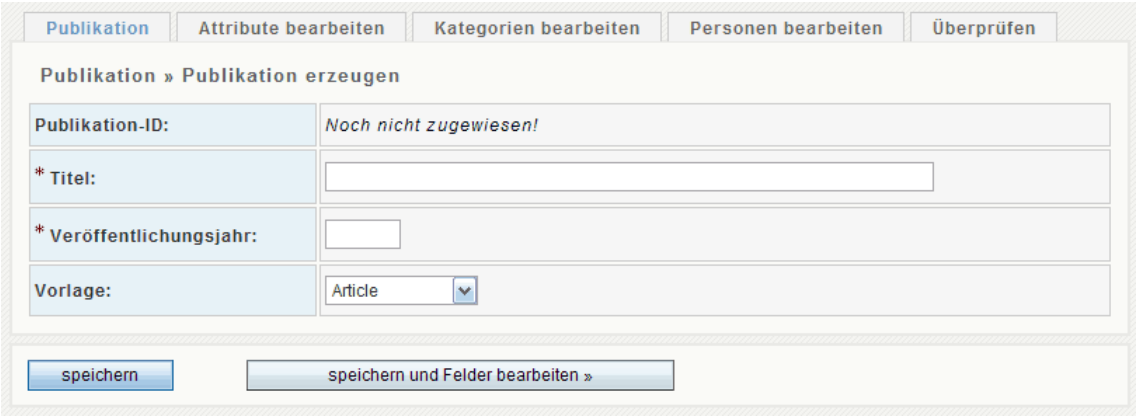


Abbildung 11.4.: Publikation anlegen

Die weiteren Schritte sind in beliebiger Reihenfolge durchführbar.

Felder bearbeiten: Hier werden die von der Vorlage vorgegebenen Felder ausgefüllt. Darüber hinaus können weitere publikationsspezifische Attribute hinzugefügt werden.

Kategorien bearbeiten: Hier können zu einer Publikation Kategorien hinzugefügt oder entfernt werden. Das Hinzufügen von Kategorien geschieht entweder anhand des Kategorie-Browsers oder anhand der Kategoriekürzel. Darüber hinaus können alle Kategorien einer vorhandenen Publikation direkt importieren werden.

Autoren bearbeiten: Um einer Publikation Autoren zuzuweisen oder bereits zugewiesene Autoren wieder zu entfernen, wird diese Seite verwendet. Auch hier können Autoren von einer bestimmten bereits bestehenden Publikation importiert werden. Siehe Abbildung 11.5.

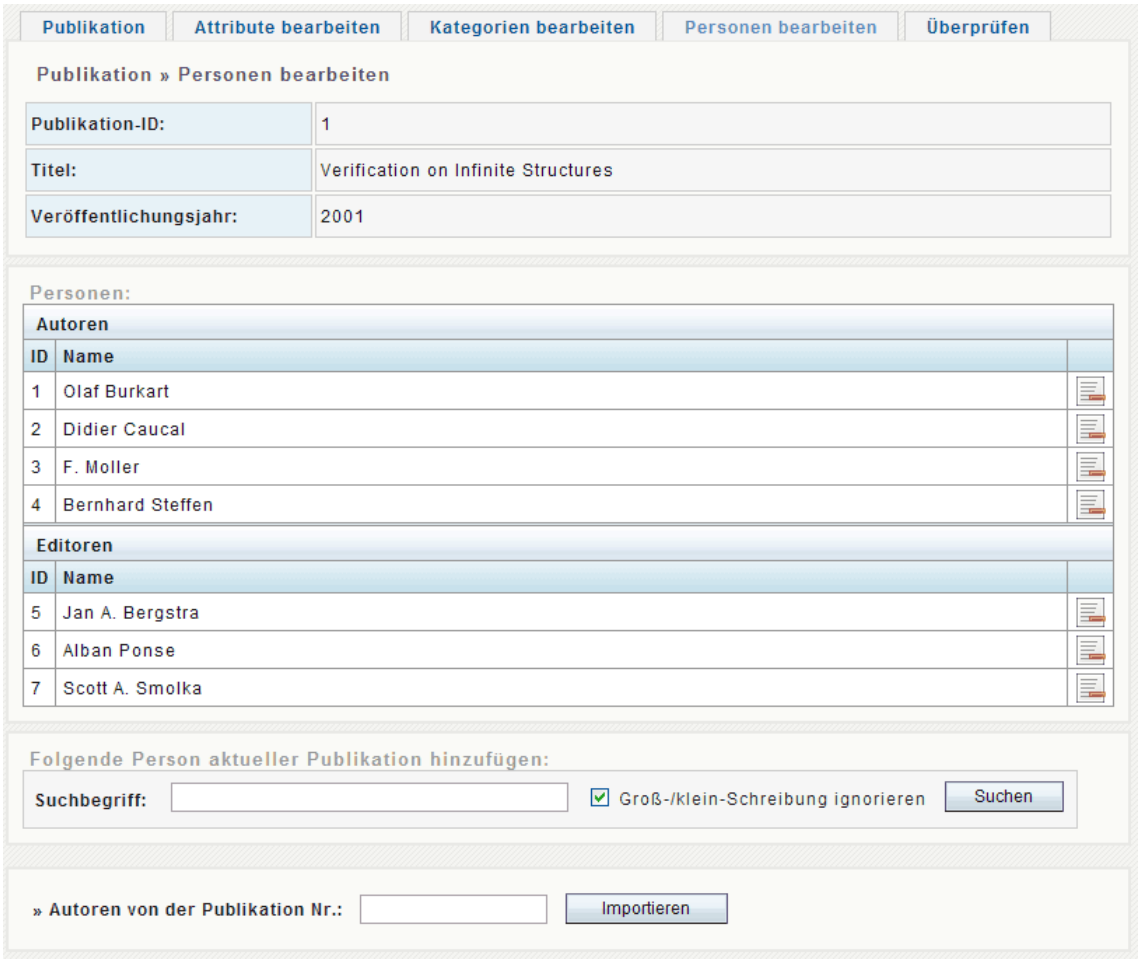


Abbildung 11.5.: Autoren bearbeiten

Überprüfen: Zuletzt erfolgt eine Überprüfung der eingegebenen Daten. Außerdem kann hier der Status der Publikation geändert werden. Einen Überblick über die möglichen Zustände einer Publikation ist in Kapitel 5 auf Seite 38 zu finden. Siehe Abbildung 11.6.

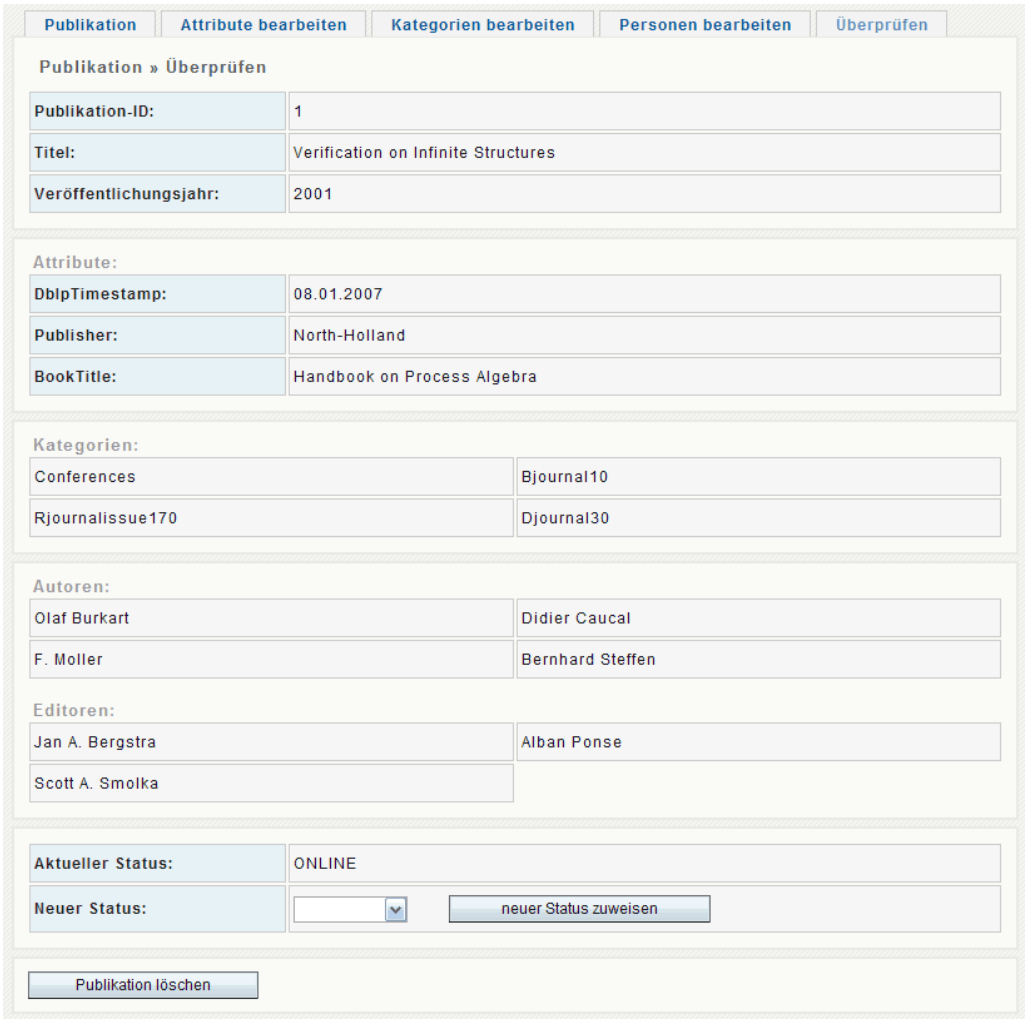


Abbildung 11.6.: Überprüfen

Attribut-Pool und Vorlagen

Im Attribut-Pool können Attribute erzeugt werden. Die Attribute können dann unter dem Menüpunkt *Vorlagen* zu einer oder mehreren Vorlagen zugeordnet werden.

11.4.3. Autoren verwalten

Der Autorenbereich enthält zwei Unterseiten zum Durchsuchen nach Autoren und zum Anlegen neuer Autoren.

Für die Suche nach Autoren existieren ein Formular Expertensuche. (Siehe Abbildungen 11.7) Diese Suchen verwenden die Funktionen des *Webclienten* (siehe hierzu auch Kapitel 8 auf Seite 60 und Kapitel 9 auf Seite 76).

Der Abschnitt *neuen Autoren hinzufügen* unterteilt sich in drei Unterabschnitte.

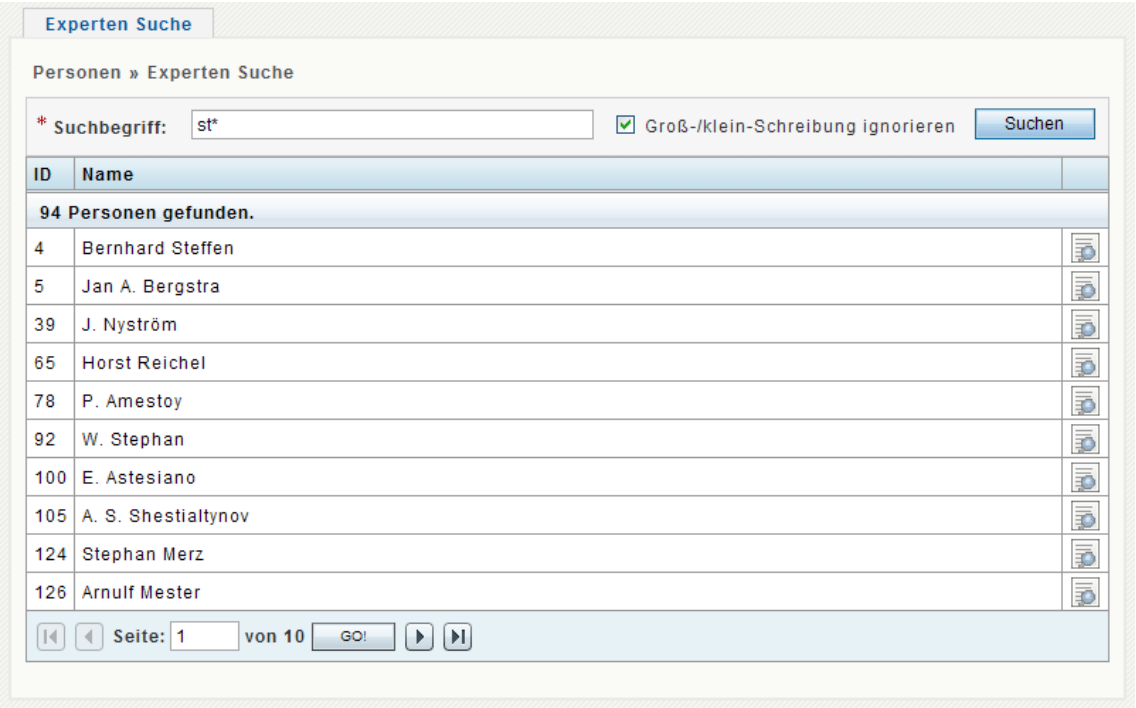


Abbildung 11.7.: Die Expertensuche

Der erste Abschnitt *neuen Autor anlegen* enthält drei Pflichtfelder und muss immer zuerst durchgeführt werden. Danach können die anderen Abschnitte in beliebiger Reihenfolge bearbeitet werden.

Im zweiten Abschnitt können einem Autor weitere Namen zugeordnet sowie bestehende Namen bearbeitet oder entfernt werden. Siehe Abbildung 11.8.

Im letzten Abschnitt können schließlich die für einen Autor vorhandenen Datenfelder wie Homepage oder E-Mail bearbeitet werden.

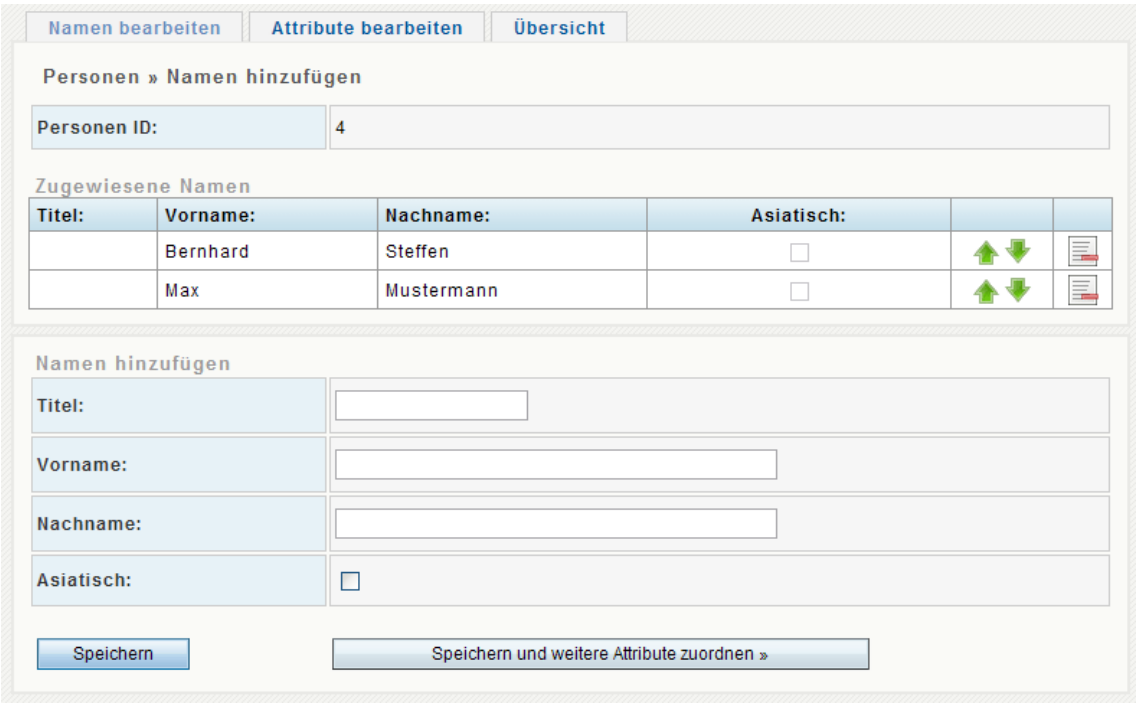


Abbildung 11.8.: Namen hinzufügen

11.4.4. Benutzerverwaltung

Die Benutzervervaltung ermöglicht es, Benutzer und Rollen zu verwalten. Diese Option ist nur Benutzern sichtbar, die selbst Rechte zum Anlegen/Löschen/Modifizieren von Benutzern und/oder die entsprechende Rollen besitzen.

Benutzer und Rollenverwaltung

Innerhalb des Menüpunktes *Benutzerverwaltung* ist es möglich, beliebige Benutzer anzulegen und diesen Rollen zuzuweisen.

Die *Rollenverwaltung* ermöglicht es, beliebige Rollen anzulegen und diesen dann eine Menge von definierten Zugriffsrechten zuzuweisen. Die Seite selbst ist wiederum dynamisch. In Abhängigkeit der Zugriffsrechte des aktuellen Benutzers werden die vom System angebotenen Funktionen angezeigt, ausgeblendet oder deaktiviert.

Teil III.

Schlussbemerkungen

12. Zusammenfassung

Das Softwaresystem oCOBS stellt eine Weiterentwicklung der an der Universität Trier betriebenen Literaturdatenbank DBLP dar. Wie viele andere Softwareprojekte, muss auch die DBLP mit begrenzten Ressourcen umgehen. Hierbei entsteht ein Trade-off zwischen der Pflege und Wartung der Bibliographiedaten selbst, und der Weiterentwicklung der Webanwendung an sich sowie ihrer Funktionalität (siehe hierzu auch 1.1 auf Seite 2). Bei der Neuentwicklung des Systems wurde großer Wert auf Produktivität, Zuverlässigkeit, Erweiterbarkeit und Skalierbarkeit sowie Zukunftssicherheit gelegt. Erwähnenswert ist an dieser Stelle die Verwendung des am Lehrstuhl 5 entwickelten Konzepts der Lightwight Process Coordination (siehe hierzu Kapitel 3.7 auf Seite 27), welche uns diese Grundsätze sichert. Auch bei der Auswahl der Technologien müssen die angestrebten Grundsätze berücksichtigt werden. Werkzeuge wie das JavaABC tragen spürbar zur Produktivität bei. So wurden zum Beispiel große Teile des Webclients mit Hilfe des JavaABC modelliert und generiert (Auflistung aller Einsatzgebiete des JavaABC innerhalb der Entwicklung siehe Kapitel 3.7 auf Seite 27).

Die Entwicklung des oCOBS-Systems ist in drei Aufgabenbereiche gegliedert. Das Datenmodell stellt einen Speicherort für die Daten bereit. Eine Administrationsschnittstelle dient zur Pflege und Aktualisierung der Daten. Schließlich stellt der Webclient eine webbasierte Schnittstelle für die Benutzer zu dem System dar.

Im Unterschied zu dem DBLP-System verwendet das neue oCOBS-System keine Textdateien zur Speicherung der Informationen, sondern verwahrt die bibliographischen Daten in einem relationalen Datenbanksystem. Dank Hibernate setzt das System keine spezielle Datenbank eines bestimmten Herstellers voraus. Viele der auf dem Markt erhältlichen relationalen Datenbanksysteme können als Datenquelle dienen (siehe hierzu Kapitel 5 auf Seite 38).

Die Administrationsschnittstelle stellt ein webbasiertes System zur Pflege und Aktualisierung der Daten dar. Dieses System basiert auf einem Rechte-Management-System und ermöglicht ein rollenbasiertes Arbeiten (siehe hierzu Kapitel 11 auf Seite 90).

Das Frontend des oCOBS bildet ein Webclient. Dieses System ist ebenfalls webbasiert und bietet verschiedene Möglichkeiten, nach Publikationen und Autoren zu suchen. Der Benutzer kann hier entweder sehr simpel die Datenbank nach einem oder mehreren Begriffen durchsuchen aber auch komplexe Suchanfragen starten (siehe hierzu Kapitel 8 auf Seite 60). Aus diesem Grund wurde eigens eine Anfragesprache entwickelt (siehe hierzu Kapitel 9 auf Seite 76). Außerdem gibt es im Webclient

die Möglichkeit Publikationen in einer Art Warenkorb, dem sogenannten Merkzettel, zusammenzustellen, um diese zu einem späteren Zeitpunkt einzeln oder in der Gesamtheit zu exportieren (siehe hierzu Kapitel 8.1.6 auf Seite 65).

13. Ausblick

Zum Abschluss liefert dieses Kapitel einen Überblick der möglichen Erweiterungen des Projekts. Die folgenden Kapitel stellen einige Ideen zur Verbesserung des Web- und Adminclients sowie des Datenmodells und des Imports im Einzelnen vor.

13.1. Ausblick für die Suchfunktion

Durch die Umstellung von Criteria auf SQL ist die Criteria-Komponente nicht weiterentwickelt worden. Mit Hilfe von Bibliotheken zur Volltextsuche in Java, wie zum Beispiel *Lucene* (siehe [4]), könnte eine Implementierung mit Criteria oder *HQL* die Flexibilität und Geschwindigkeit der derzeitigen Lösung mit den Vorteilen einer Abstraktionsschicht, die Hibernate darstellt, vereinen.

Derzeit bietet oCOBS lediglich die Möglichkeit der Suche nach Publikationen und Autoren. Eine Anfrage nach Kategorien wäre eine sinnvolle Erweiterung.

Der nachfolgenden Projektgruppe wird eventuell Zugriff auf den Volltext vieler Dokumente zur Verfügung stehen. Eine dahingehende Zusammenarbeit mit den Verlagen wird vom Lehrstuhl angestrebt. Für die Suchkomponente ergibt sich hier die Möglichkeit, neben den Metadaten auch den Volltext zu durchsuchen.

Weiterhin sind einige Anfragen, wie zum Beispiel ungeschachtelte Ausdrücke der einfachen Suche (siehe Abschnitt „Expertensuche“ auf Seite 82), so generisch, dass sie in der SQL-Komponente mit *Prepared Statements* beschleunigt werden können. Die `UNION`-Befehle für die Autor- und Editor-Beziehungen (siehe Abschnitt „Bibliographie“ auf Seite 38) sind für komplizierte Ausdrücke mehrfach auszuführen. Temporäre Tabellen würden die Performanz solcher Anfragen verbessern. Ferner sind die Möglichkeiten, mit Hilfe von Optionen in den `AdvancedOptions` die Geschwindigkeit der Suchanfragen zu verbessern, noch nicht ausgeschöpft. Editoren bilden einen häufig vernachlässigbaren Teil des Datenbestandes, da zumeist nach den Verfassern beziehungsweise den Publikationen eines Verfassers gesucht wird. Daher wäre in einigen Fällen eine signifikante Senkung der Antwortzeit einer Suchanfrage möglich, wenn die `UNION`-Ausdrücke weggelassen würden.

Empfohlen wird des weiteren eine Trennung der Entitäten in einzelne Klassen, statt der verwendeten `enum` mit den Werten `PUBLICATION` und `AUTHOR`. Dadurch können zahlreiche Fallunterscheidungen vermieden werden und durch die Verwendung einer abstrakten Klasse kann mittels Vererbung doppelter Quellcode vermieden werden. Dies ist insbesondere sinnvoll, wenn die Suchfunktion um Kategorien

erweitert wird.

Im Ausblick des Zwischenberichts war angedacht, den Durchlauf des *Abstract Syntax Trees* – kurz AST – mit Hilfe des JavaABCs zu modellieren. Durch die Umstellung auf SQL und Tsearch2 ist dies zeitlich nicht mehr möglich gewesen.

13.2. Webclient

Fehler in den Daten einer Publikation lassen sich nie ganz ausschließen. Eine sinnvolle Erweiterung der Anwendung wäre daher ein Mechanismus, durch den der Benutzer den Administrator auf Fehler aufmerksam machen kann. Vorstellbar ist die Bereitstellung eines Formulars, durch das der Benutzer den fehlerhaften Eintrag angeben kann. Nachdem dieses übermittelt wurde, muss eine manuelle Überprüfung der bisher gespeicherten Daten durch den Administrator stattfinden. Anschließend kann gegebenenfalls eine Korrektur erfolgen und der Benutzer per E-Mail über die durchgeführten Änderungen informiert werden.

Weitere Verbesserungen sind bei der Handhabung der Bibliographie möglich. Insbesondere könnte die Filterung von *rootcategories* verbessert werden, indem zum Beispiel nicht mehr zwischen Groß- und Kleinschreibung unterschieden wird und Sonderzeichen unter einem gemeinsamen Punkt, beispielsweise „andere“, zusammengefasst werden.

Auch der Export kann um weitere Funktionen und Formate ergänzt werden. Dazu gehört der Export ganzer Sessions, Abschnitte, Hefte, Bände, Zeitschriften oder Tagungsserien. Nützliche neue Formate sind möglicherweise verbreitete Bibliographie-Formate wie Endnotes, Dublin-Core[1] oder OAI[2].

Eine wesentliche Verbesserung der Performanz der Applikation ist durch die Entwicklung eines Verfahrens zum Caching häufig genutzer Seiten zu erwarten. Dabei muss die Konsistenz sichergestellt werden. Das lässt sich zum Beispiel durch die Verwendung von Zeitstempeln realisieren, die im Falle eines Cache-Treffers mit in der Datenbank vorhandenen verglichen werden. Die Seite muss nur dann neu aufgebaut werden, wenn der Zeitstempel in der Datenbank neuer ist. Ansonsten kann die angeforderte Seite sehr schnell aus dem Cache geladen werden. Erschwert wird die Konsistenzhaltung unter anderem dadurch, dass Änderungen einer Publikation weitreichende Konsequenzen haben können, da sich eine einzelne Änderung auf mehrere Autorenseiten auswirken kann.

[1] siehe `http://dublincore.org/`
[2] siehe `http://www.openarchives.org/`

13.3. Datenmodell

Insbesondere beim Import des bibliographischen Hypertextes ist deutlich geworden, dass sich gewisse Informationen aus der DBLP nur schwer oder überhaupt nicht in das Kategorieverzeichnis von oCOBS übertragen lassen. So konnten etwa Angaben zu VLDB Awards[3] oder Komitee Satzungen[4] nicht importiert werden, um konkrete Beispiele zu nennen.

Ebenso lassen sich anwendungsinterne Querverweise auf andere Kategorien oder Publikationen nicht außerhalb der im Datenmodell explizit modellierten Assoziationen darstellen. Hierdurch bedingt konnten etwa die in Hinweisen auf Nachfolgerkonferenzen[5] enthaltenen Hyperlinks nicht importiert werden.

Weiterhin erweist sich die aktuelle Modellierung der Eltern-Kind-Beziehung zwischen Kategorien als unbefriedigend. Bislang kann bei der Anzeige dieser Assoziation – etwa mittels Hyperlink – der Assoziationspartner lediglich durch den Titel der betreffenden Kategorie kenntlich gemacht werden. Im Hinblick auf Flexibilität ist hingegen wünschenswert, dass zu der Assoziation selbst eine (optionale) Beschriftung angegeben werden kann. Hierdurch könnte die Anzeige kontextabhängig gestaltet werden, um Verbesserungen in der Navigation zu ermöglichen. Konkret ließe sich so etwa ein Index aller Zeitschriften realisieren, in dem eine Zeitschrift wiederholt aber unter unterschiedlichen Schlagworten aufgeführt wird.

Im Zuge zukünftiger Weiterentwicklungen der Anwendung, deren Nutzungsschwerpunkt insbesondere das hypermediale Recherchieren von Literatur ist, sollte daher nach Möglichkeiten zur Behebung der oben geschilderten Schwächen gesucht werden. Ein Lösungsansatz hierzu könnte die Erweiterung von Kategorien um ein Feld zur Aufnahme von beliebig langem und (eingeschränkt) formatierbarem Hypertext sein, um so teilweise die Ausdrucksstärke des in der DBLP eingesetzten bibliographischen Hypertextes nachzuempfinden.

13.4. Import

Da beim Import beziehungsweise bei der Erstellung neuer Publikationen bisher nicht überprüft wird, ob bereits ein ähnlicher Datensatz vorhanden ist, der sich auf die gleiche Publikation bezieht, lassen sich Dubletten bisher nur durch manuelle Suche erkennen.

Insbesondere aufgrund der geplanten Nutzung mehrerer Datenquellen mit sich überschneidenden Datenbeständen erscheint es daher zur Erhöhung der Datenqaulitität als sinnvolle Erweiterung, einen Dienst zu implementieren, der auf der Basis eines geeigneten Ähnlichkeitsmaßes Dubletten erkennt und dem Administrator mel-

[3] siehe `http://www.informatik.uni-trier.de/~ley/db/conf/vldb/award1999.html`
[4] siehe `http://www.informatik.uni-trier.de/~ley/db/conf/dl/charter.html`
[5] siehe `http://www.informatik.uni-trier.de/~ley/db/conf/etaps/index.html`

det, beziehungsweise bei entsprechend hoher Übereinstimmung automatisch miteinander verschmilzt.

Ein weiterer Ansatzpunkt zur Erhöhung des Automatisierungsgrades ergibt sich beim inkrementellen Import des bibliographischen Hypertexts. Während bei dem Import der bibliographischen Datensätze aufgrund der hohen Datenqualität der DBLP eher selten Änderungen an bereits importierten Publikationen zu erwarten sind, stellen Modifikationen eines bereits importierten Eintrags des Hypertextes einen regelmäßigen Anwendungsfall dar, beispielsweise bei dem Hinzufügen eines neuen Volumes zu einem Journal oder bei der Aufnahme einer neuen Konferenz-Reihe.

Bisher wird ein derartiges Fehlschlagen des Prüfsummenvergleichs beim inkrementellen Import des Hypertextes als Folge der Modifikation von einzelnen Hypertext-Einträgen als Task an den Administrator gemeldet. Jedoch erscheint es als Weiterentwicklung denkbar, dass entweder die aktuellen XML-Daten mit denen des letzten Import-Vorgang direkt verglichen werden oder aber dass die im Rahmen der Verarbeitung eines Hypertext-Eintrags erzeugten Kategorien und deren Attribute mit den Resultaten des vorherigen Import-Prozesses verglichen werden.

Darauf aufbauend könnte eine automatisierte Verarbeitung dieser Modifikationen realisiert werden.

13.5. Adminclient

Die Administrationsoberfläche bietet viele Erweiterungspunkte. Zu einem sollte der Funktionsumfang der Administrationsoberfläche die Möglichkeiten des Datenmodells voll ausschöpfen. So ist es bei den Autoren und Kategorien bislang nur möglich, die Standardattribute zu bearbeiten. Es sollte angestrebt werden, das Vorlagenkonzept auch auf die Autoren und Kategorien anzuwenden. Weiterhin könnten Statistiken angezeigt und gegebenenfalls auch editiert werden. Hierzu ist aber auch eine Erweiterung der Webservices nötig. Als zweiter Punkt sind optionale Features zu nennen. So könnte die Startseite personalisiert werden, indem dem Benutzer die Möglichkeit geboten wird, die Startseite selbst zu konfigurieren. Dieses beinhaltet sowohl die Auswahl an Serverstatistiken als auch die Anzahl und die Art der Suchfilter selbst zu bestimmen, die einen schnellen Zugriff auf zuletzt bearbeitete Entitäten ermöglichen.

Index

Index

Literaturverzeichnis

[1] ACM SIGMOD = Association for Computing Machinery Special Interest Group Management of Data : *Homepage*. http://www.sigmod.org/.

[2] Apache Software Foundation: *Axis*. http://ws.apache.org/axis/.

[3] Apache Software Foundation: *Axis 2*. http://ws.apache.org/axis2/.

[4] Apache Software Foundation: *Apache Lucene*. http://lucene.apache.org/java/docs/index.html, 2006.

[5] Bahlo, Tim: *Seminarthema 1: DBLP*. PG L2EE Seminar, 30 März 2006.

[6] Bartunov, Oleg und Teodor Sigaev: *Tsearch2 – full text extension for PostgreSQL*. http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/.

[7] Christian Winkler: *DBSchema*. http://jabc.cs.uni-dortmund.de:8002/plugins/dbschema/index_en.html.

[8] FIZ Karlsruhe Informationsdienste: CompuScience: *Homepage*. http://www.fiz-informationsdienste.de/de/DB/compusci/.

[9] Güting, Ralf Hartmut und Martin Erwig: *Übersetzerbau*. Springer, 1999.

[10] Hibernate: *Hibernate Reference Documentation*. http://www.hibernate.org/hib_docs/v3/reference/en/html/.

[11] JavaCC: *Homepage*. https://javacc.dev.java.net/.

[12] Java Community Process: *JSR 173: Streaming API for XML*. http://jcp.org/en/jsr/detail?id=173.

[13] Java Community Process: *JSR 181: Web Services Metadata for the JavaTM Platform*. http://jcp.org/en/jsr/detail?id=181.

[14] JUnit: *Homepage*. http://www.junit.org/.

[15] Ley, Michael: *XML Dateien der DBLP*. http://dblp.uni-trier.de/xml/.

[16] Ley, Michael: *Die Trierer Informatik-Bibliographie DBLP*. In: *GI Jahrestagung*, Seiten 257–266, 1997.

[17] Ley, Michael: *Slides of: The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives.* In: *SPIRE*, 2002.

[18] Marco Bakera, Clemens Renner: *GEAR*. http://jabc.cs.uni-dortmund.de:8002/plugins/gear_en.html.

[19] Microsoft Bay Area Research Center: *Homepage*. http://research.microsoft.com/.

[20] Minter, Dave und Jeff Linwood: *Pro Hibernate 3*. Apress, 1. Auflage, 2005.

[21] Müller-Olm, Markus, David Schmidt und Bernhard Steffen: *Model-Checking – A Tutorial Introduction*. In: *Static Analysis, 6th International Symposium, SAS '99, Proceedings*, Band 1694 der Reihe *Lecture Notes in Computer Science*, Seiten 330–354, Venice, Italy, September 1999. Springer-Verlag GmbH. http://link.springer.de/link/service/series/0558/bibs/1694/16940330.htm.

[22] Nagel, Ralf und others: *Java ABC Framework Homepage*. http://www.jabc.de.

[23] Neubauer, Johannes: *Seminarthema 3: JavaABC (Konzepte, Architektur und Standard-Plugins)*. PG L2EE Seminar, 30 März 2006.

[24] Projektgruppe 494 Universität Dortmund: *Einführung in die Suchfunktion von oDOBS*. http://odobs.cs.uni-dortmund.de/wiki/index.php/Tutorial_Suche, Januar 2007.

[25] Sun Microsystems: *The Reflection API*. http://java.sun.com/docs/books/tutorial/reflect/.

[26] Sven Jörges: *Formula Builder*. http://jabc.cs.uni-dortmund.de:8002/plugins/formulabuilder_en.html.

[27] Sven Jörges: *Genesys*. http://jabc.cs.uni-dortmund.de:8002/plugins/codegenerator_en.html.

[28] TestNG: *Homepage*. http://testng.org/.

[29] Vollmer, Stephan: *Portierung des DBLP-Systems auf ein relationales Datenbanksystem und Evaluation der Performance*. Diplomarbeit, Lehrstuhl für Datenbanken und Informationssysteme, Universität Trier, März 2006. `http://dbis.uni-trier.de/Diplomanden/Vollmer/vollmer.shtml`.

[30] W3C: *XML Path Language*. `http://www.w3.org/TR/xpath`.

[31] W3C: *Web Services Description Language (WSDL) 1.1*. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, 2001.

[32] W3C: *SOAP Version 1.2 Part 1: Messaging Framework*. `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`, 2003.

[33] Wikipedia: *Personal name*. `http://en.wikipedia.org/wiki/Personal_name`.

[34] Wikipedia: *Service Oriented Architecture*. `http://de.wikipedia.org/wiki/Serviceorientierte_Architektur`.

[35] Witten, Ian H., Alistair Moffat und Timothy C. Bell: *Managing Gigabytes: Compressing and Indexing Documents and Images*. `http://www.cs.mu.oz.au/mg/`.